80/263/CDV

COMMITTEE DRAFT FOR VOTE (CDV)
PROJET DE COMITÉ POUR VOTE (CDV)

	Project number Numéro de projet 80/61162-420/Ed. 1	
IEC/TC or SC: **80** CEI/CE ou SC:	Date of circulation Date de diffusion **2000-04-07**	Closing date for voting (Voting mandatory for P-members) Date de clôture du vote (Vote obligatoire pour les membres (P)) **2000-09-15**
Titre du CE/SC:	TC/SC Title: Maritime navigation and radiocommunication equipment and systems	
Secretary: M. A. RAMBAUT - United Kingdom Secrétaire:		
Also of interest to the following committees Intéresse également les comités suivants	Supersedes document Remplace le document 80/175/CD & 80/176/CD	
Horizontal functions concerned Fonctions horizontales concernées ☐ Safety / Sécurité ☐ EMC / CEM ☐ Environment / Environnement ☐ Quality assurance / Assurance qualité		

Titre :

Title :

Maritime navigation and radiocommunication equipment and systems - Digital interfaces - Part 420: Companion standard requirements and basic companion standards - Multiple talker and multiple listeners - Ship systems interconnection

Introductory note

IEC 61162-4 Series specifies a communication protocol for use in integrated systems. It defines a ship wide and system level integration mechanism that complements communication solutions provided by other parts of the IEC 61162 series. It is also expected that the IEC 61162-4 Series will be used for data acquisition by higher level, non real-time and non-critical administrative workstations and personal computers. IEC 61162-4 Series has been developed as a network that can support a high number of nodes (several hundred if proper segmentation is used), with response times between 0.1 second and 1 second dependent on load. Ethernet and Internet protocols are employed at the transport level.

IEC 61162-4 has been divided into four different parts numbered IEC 61162-400, 401, 410 and 420.

ATTENTION **Parallel IEC CDV/CENELEC Enquiry)**	**ATTENTION** **CDV soumis en parallèle au vote (CEI) et à l'enquête (CENELEC)**

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

MARITIME NAVIGATION AND RADIOCOMMUNICATION EQUIPMENT AND SYSTEMS- DIGITAL INTERFACES-

Part 420: Companion Standard Requirements and Basic Companion Standards - Multiple Talker and Multiple Listeners –Ship Systems Interconnection.

FOREWORD

1) The IEC (International Electrotechnical Commission) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of the IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, the IEC publishes International Standards. Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. The IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of the IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested National Committees.
3) The documents produced have the form of recommendations for international use and are published in the form of standards, technical specifications, technical reports or guides and they are accepted by the National Committees in that sense.
4) In order to promote international unification, IEC National Committees undertake to apply IEC International Standards transparently to the maximum extent possible in their national and regional standards. Any divergence between the IEC Standard and the corresponding national or regional standard shall be clearly indicated in the latter.
5) The IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with one of its standards.
6) Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. The IEC shall not be held responsible for identifying any or all such patent rights.

This CDV for the The International Standard IEC 61162-420 has been prepared by Technical Committee 80: Maritime Navigation and Radiocommunication Equipment and Systems.

[This CDV is proposed to cancel and replace the first edition of the CD]

The text of this standard is based on the following documents:

FDIS	Report on voting
XX/XX/FDIS	XX/XX/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 3.

The committee has decided that the contents of this publication will remain unchanged until April 2003. At this date, the publication will be

- reconfirmed;
- withdrawn;
- replaced by a revised edition, or
- amended.

INTRODUCTION

The International Standards IEC 61162-series have been prepared by Technical Committee 80: Maritime Navigation and Radiocommunication Equipment and Systems.

IEC 61162 is a four part standard which specifies four digital interfaces for applications in marine navigation, radio-communication and system integration.

The 4 parts are:

IEC 61162-1 Single Talker and Multiple Listeners

IEC 61162-2 Single Talker and Multiple Listeners - High Speed Transmission

IEC 61162-3 Multiple Talker and Multiple Listeners - Serial Data Instrument Network

IEC 61162-4 Multiple Talker and Multiple Listeners - Ship Systems Interconnection. This part is sub-divided into a number of individual standards with part numbers in the 400 series.

This part of the standard contains the specification of a description language for IEC 61162-4 series companion standards (user layer specifications), a framework for the organisation of such companion standard descriptions and also the descriptions of basic components that can be used as a starting point to build IEC 61162-4 series components and networks.

Later standards in the companion standard series (IEC 61162-42x) are expected to address more concrete interface requirements for specific navigational equipment.

CONTENTS

1 Scope

1.1 Contents of this document

This document contains the specification of the requirement for and basic components of the IEC 61162-4 series Companion Standards. In the following, these components are referred to as follows:

a) **PCS** (PISCES Companion Standards) which contains the rules for creation of companion standards. The general principles underlying the PCS are described in clause 4.

b) **PCSDL** (PCS Description Language). Part of the PCS is the definition of the syntax for the various types of companion standard documents that make them readable by computer tools. The PCSDL is described in clause The Companion Standard Reference.

c) **Function block** description. The function block description is a high level and graphical description of applications using the IEC 61162-4 series interface standard. The function block syntax is specified in clause 7.

d) **PFS** (PISCES Foundation Specifications) which contain a framework for classification of applications adhering to the IEC 61162-4 standard. The PFS will also provide a minimum level of interoperability between different manufacturers' applications using this framework. The PFS is described in clause 6.

Clause 5 contains the complete reference to the PCS description language. Clause 5.2 explains the basic concept of the PCS which is given by the distinction between four types of specifications: applications, interfaces, information and data types. General conventions with respect to the syntax of the PCS can be found in 5.3. All PCS documents are based on a similar structure. This approach is intended to make it easier to become familiar with the syntax and semantic of the PCS which is defined in 5.3.1. The four clauses thereafter explain in detail the syntax and semantic of the four different types of specifications generated by the PCS.

Clause 6 describes the relationship between the different classes of IEC 61162-4 applications and gives an overview of their functionality. Annexes contain the detailed PCS definitions for the classes.

1.2 The Purpose of Companion Standards

The objective of companion standards is to provide definitions of the information that is transferred within an integrated ship control systems and of how these information items can be accessed or provided. Further more, the standard shall allow the definition of the actual network interfaces the applications use to connect to the system. The description format is machine-readable, allowing an automatic compilation of the description into interface software.

A companion standard allows the reader to at will shift the focus between a technical specification and a definition of interfaces and information items. The development team can determine information attributes like unit, power, accuracy and the structure of the system architecture and create a common interpretation basis for data before the system implementation. The formalisms underlying the specification language will at the same time provide an unambiguous and precise description of the equipment interfaces which allow the use of computer tools to automatically generate interface program code or to inspect and manipulate interfaces on-line, e.g., for debugging and monitoring purposes.

2 Normative references

All normative references defined in IEC 61162-400 are also normative in this part of the standard.

The following normative documents contain provisions, which through reference in the text, constitute provisions of this International Standard. At the time of publication, the editions

indicated were valid. All normative documents are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. Members of IEC and ISO maintain registers of currently valid International Standards

IEC 61162-400: (to be published), Maritime navigation and radio communication equipment and systems - Digital interfaces - Part 400: Multiple talker and multiple listeners - Ship Systems Interconnection – Introduction and General Principles

3 Definitions

For the purpose of this International Standard, the following definitions apply

3.1 Terms and abbreviations

Abstract specification
PCS specifications that are part of the PFS (defined in this documen) are not intended for direct implementation and is termed “abstract”.

Application interface
A collection of interface components instantiated in an application definition document as one protocol level interface

CP – Connection Point
An application interface consists of a number of individual “functions” that can be called into or out from. Each of these functions, as it references a remote entity is called a “connection point”.

CS – Companion standard
A protocol layer on top of the normal OSI application level (see definition of companion standard in [IEC 61162-400]), representing the definition of how the application layer functionality is used to implement a certain application’s interface functionality. Also called user layer.

Function block
A high level, partly graphical representation of an application’s place in an integrated system. The function block presents all interfaces and relationship between these on an overview level.

Interface (component)
An interface component is a collection of connection points (CP) in one INTERFACE definition document. This interface component can be aggregated with other interface components into an actual interface as defined in the A-profile. The actual interface (A-profile sense) is defined as the application interface in the APPLICATION document.

PCS - PISCES Companion Standard
The complete concept of companion standards, including description language (PCSDL), function blocks and the foundation classes (PFS).

PCSDL - PISCES Companion Standard Description Language
The formal interface description language for PCS.

PFS - PISCES Foundation Specifications
The interface base classes for all applications created in the framework of the PCS.

Tangible
A specification of an entity that shall be implemented (instantiated) at some time.

3.2 General Typographical Rules in this Document

The following typographical rules apply throughout this document:

a) Fragments and complete pieces of PCSDL source code is written in `Courier`

b) Tokens written in capitals, typeset in `COURIER` are reserved PCSDL keywords.

c) Words in pointed brackets, '<' and '>', define place-holders that have to be filled with the appropriate token as described in the text.

d) Tokens in square brackets, '[' and ']', define tokens that are optional, e.g. parts of a statement that is only required under special circumstances.

e) Ellipses, . . ., show that the preceding item can be repeated.

Clause 5.3 defines other typographical and lexical conventions that apply to PCSDL documents.

4 General Principles for the PCS

4.1 General Structure

4.1.1 Purpose

The main purpose of the PCS is to give an unambiguous way to interpret data transmitted via the IEC 61162-401 A-profile protocol. In this sense, the companion standard adds meaning to the data transmitted via the protocol, converts it to information and makes it usable for application modules connected to the network. To serve this purpose, the PCS shall provide the following:

a) Establish a language to define information types, application interfaces and applications. This language has to be human readable as well as interpretable by a computer. It is called PISCES Companion Standard Description Language (**PCSDL**).

b) Provide a standardised set of information types and interfaces which can be used as a basis to create customised (i.e., vendor specific) application and interface descriptions. This set is also called the PISCES Foundation Specification (PFS, see 4.4).

c) Provide a general framework for a high level description of applications that use the IEC 61162-4 standard for communication. This is the function block specification format.

4.1.2 Components of the PCSDL

The PCSDL supports the generation of four different types of specifications as outlined in the following clauses.

The three first document types can be used to generate protocol (A-profile) related entities, e.g., data object names, MAU names and format strings. The information specification can be used to add more application related meaning to the information entities. The information specifications can also use an extended format string syntax to implement higher level functionality based on the A-profile specification.

4.1.2.1 Application

Representation of application units within the PCS. An application is defined by application interfaces specifying the respective inputs and/or outputs. The application specification can be

looked at as the specification of how one particular piece of equipment is connected to the system. Applications will normally consist of a number of *interfaces* configured as either providing data to or using data from the system.

4.1.2.2 Interface (component)

Specification of connection mechanisms used between applications. Each interface has one or more connection points (CP). Each connection point specify one or more information types received or provided by the respective interface. The connection point specification consists of the data structures transferred, the information the data structures carry and how the information can be accessed (read, write etc.).

4.1.2.3 Data type

Data type definition specifies the structure and to some degree the content of data transmitted via the PISCES network. This document can be optionally exchanged or complemented with *information specifications*.

4.1.2.4 Information

An information specification represents and defines the content and interpretation of data transmitted via the PISCES network. The purpose of these specifications is to define the format of exchanged data and specify the usage of the respective information. This type of document extends and replaces the function of the data type definition by making it possible to give more meaning to the data structures. Any information specification can be converted to a data type definition, but at a loss of information meaning.

NOTE – The information type specification is not currently used in the PFS. At this time only the data type specification is used to define structure and contents of transmitted data blocks. However, it is recommended that the information type is considered for new specifications.

4.1.3 Object based principles in the PCSDL

4.1.3.1 General

The PCSDL supports an simplified object based view on companion standard specifications. To create companion standards, the following object based principles can be applied.

All specifications used within PCS are seen as encapsulated structures defining their own scope and domain. Each specification can be used as a “building block” to create new specifications following the rules given in clause 5. Two different mechanisms are available to compose new specifications from existing ones: Specialisation and Aggregation.

4.1.3.2 Specialisation

The PCSDL gives the opportunity to derive new specifications from existing ones. The new specification (i. e. application, interface or information) inherits all properties from the old specification called the base specification). This mechanism forms a generalisation-specialisation relation between these two specifications.

4.1.3.3 Aggregation

It is possible to compose new specifications by aggregating two or more existing specifications. A complex information type can e. g. be formed by including several information specifications into a new specification. The same applies for composing complex application interfaces from one or more interfaces (see 5.6).

Specifications for applications, interfaces and information are included in three separate hierarchies, see Figure 1. Each hierarchy stems from a base specification covering properties common to each specification of the respective type (i. e. information, application or interface). An overview of the PFS hierarchy for applications is given in annex E.

4.1.3.4 Data types versus information

The current version of the PFS does not make use of interface specifications and neither theobject oriented principles inherent in that. Instead, the less advanced data type definistions are used. This is due to legacy companion standards from previous versions of this document (see MiTS in Part 400).

NOTE - This part of the standard emphasises the information type documents as it is believed that this will be the description method of choice in the future. However, the reader may want to concentrate on data type documents as they are the document type most commonly used as of this version of the standard.

4.1.4 Generic and manufacturer specific companion standards

The general approach recommended by PCS is to divide the set of companion standards into two different parts (Figure 1): Generic CS and manufacturer specific CS. Generic CS will be defined by the PFS. The PFS contains “generic” specifications for data types, information, interfaces and applications. These specifications can be used by manufacturers as a starting point to define own specifications to describe their respective equipment. This means that PFS specifications can be used as templates to create equipment specific specifications. This mechanism forms a generalisation-specialisation relation between the newly created specification and the respective specification of the PFS. It has to be noted that specifications which are part of the PFS are “abstract” in that sense that they can not be used directly to specify equipment. A tangible PCS specification of a component has always to be derived from a PFS specification.

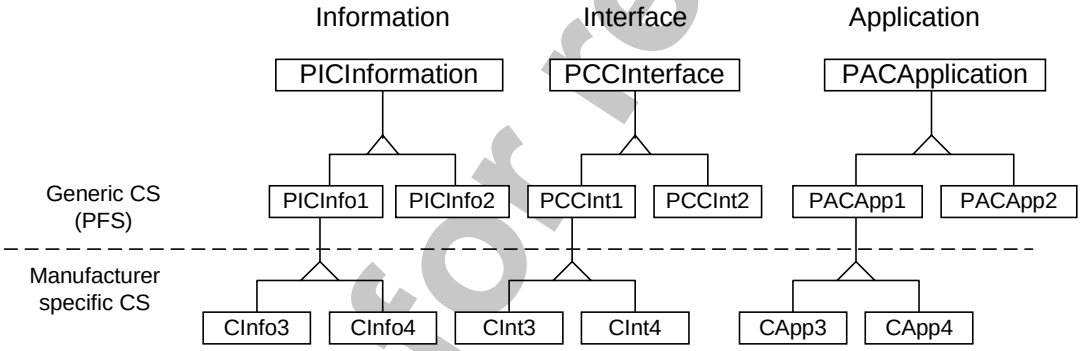

Figure 1 - General structure of PCS hierarchy of specifications

To form specifications of information, interfaces or applications, direct inheritance mechanisms can be used. The new specification inherits all properties from the specification it is derived from (the base specification). When deriving new specifications from existing ones, it is only allowed to add attributes to the newly created specifications (see also 4.2.5). Thus, the following mechanisms are the only permitted by the PCS description language:

a) Generation of a new application specification: Derivation of the new specification from an existing one (abstract or tangible specification) and addition of new application interfaces (see 5.5).

b) Generation of a new interface specification: Derivation from an existing interface specification (abstract or tangible specification) and addition of connection points (see 5.6).

c) Generation of a new information specification: Derivation from an existing specification and addition of attributes, i. e. data types or information (see 5.7).

d) Creation of new data type specifications (see 5.8).

4.1.5 Generic Companion Standards (PFS)

This part of the companion standards defines generic information or data types to be transmitted via the A-profile protocol and standardised interface or application descriptions. With respect to physical components these generic descriptions define the minimum required functionality (including the information they shall provide) of the component concerned. Generic

companion standards, including application, interface, data type or information specifications, are part of this standard and shall not be modified by manufacturers.

Thus, applications, interfaces and information specifications contained in the generic companion standard form „abstract specifications" and normally it makes little or no sense to use these directly as full specifications for actual applications. Normally, it is necessary to create a more concrete specification (a "tangible specification"), by deriving it from one of the abstract specifications.

NOTE - Generic companion standards provide the "least common denominator" for information interchange via the PISCES protocol. For instance a physical position sensor will at least own the interfaces defined for a “generic“ position sensor. In this sense these generic application and interface descriptions have to be in line with the respective IMO performance standards.

The collection of generic specifications defined by the PCS is called the PISCES Foundation Specifications (PFS).

4.1.6 Manufacturer Specific Companion Standards

To satisfy the needs of specific (i.e., physical) components, a manufacturer creates specific companion standards for information type, data types, interfaces or applications. These companion standards are derived from generic companion standards (see 4.2.7). In general, a manufacturer specific companion standard for e. g. an interface will contain the content (information types, attributes etc.) of the respective generic companion standard as a sub-set.

NOTE - The main advantage of this approach is that there is no need for a manufacturer to develop his or her specific components from scratch. This accelerates the process of specification and implementation significantly. The second advantage is that any interface derived from a generic PCS interface description provides as a minimum the information types defined within the respective specification of the PFS: even if a manufacturer specific companion standard is unknown, the services of the respective "template" can be used. For example: If manufacturer B wants to connect his ECDIS system with a GPS receiver of vendor A, the manufacturer can at least receive the information types of the generic GPS interface (i. e. position, satellites in view, valid flag and time), even if he does not know the specification of the customised interface. Only the generic interface used as the template has to be known in this case.

Companion standards for information types, interfaces and applications are defined within hierarchies (Figure 1). Information types and interfaces will be derived from the respective base specification defining the properties (attributes) common to all specifications derived from this base. There is also a base specification for applications.

The limit between the generic (normative) and the “customised” (manufacturer specific) part of the PCS is formed by a “horizontal cut” within the respective specification hierarchy. This is indicated by the dashed line in Figure 1.

Attributes and methods necessary to implement services that are needed from the A-profile will also be defined within the highest-level base specifications. They are implemented by several specialised generic interfaces. These interfaces are described within 4.4.

To provide a high level description of a new application, the manufacturer should also provide a function block specification of the application (clause 7).

4.1.7 Guidelines for using PCS

This clause gives an overview of how to use the PCSDL to create specific companion standards. The starting point is the need to connect a new application to a IEC 61162-4 compliant network. To implement such a connection, the following steps have to be carried out:

a) Identify the necessary interaction between the application concerned and other applications in the system. Add possible interactions to possible future applications where appropriate. Group these interactions roughly into draft PCS “application interfaces”. If possible, the draft should be based on an already existing similar application which is known to be conformant to the PFS.

b) Construct a function block that can be used to describe the new application. The function block should identify input interfaces, output interfaces, physical relationships between application and environment and the applications functionality.

c) If an appropriate application specification does not already exist, derive a new application specification from a PFS template (i.e., an application base specification contained in the PFS) and adapt it by adding new application interfaces.

d) Determine the number and type of each connection point in each of the application interfaces. Add new connection points to existing interfaces where appropriate. Sub-divide each application interface into separate interfaces where appropriate (see below) and create the new interface specification documents, either based on existing interface specifications, from scratch or by inheritance from interfaces in the PFS.

e) Determine input/output formats for each new connection point in the interfaces. Describe this in information and/or data type documents, where possible using inheritance from existing PCS specifications.

f) Create the final application specification document by using specialisation from the appropriate component in the PFS library and adding the newly developed or modified interfaces.

Note that **application interfaces** are defined only in application specifications. Each application interface consists of one or more interface components, grouped together to form one interface module as seen from the application. Each of the interface components is defined by one interface specification. This approach ensures high flexibility for adapting application interfaces to specific needs of applications by creating a building block system for interfaces.

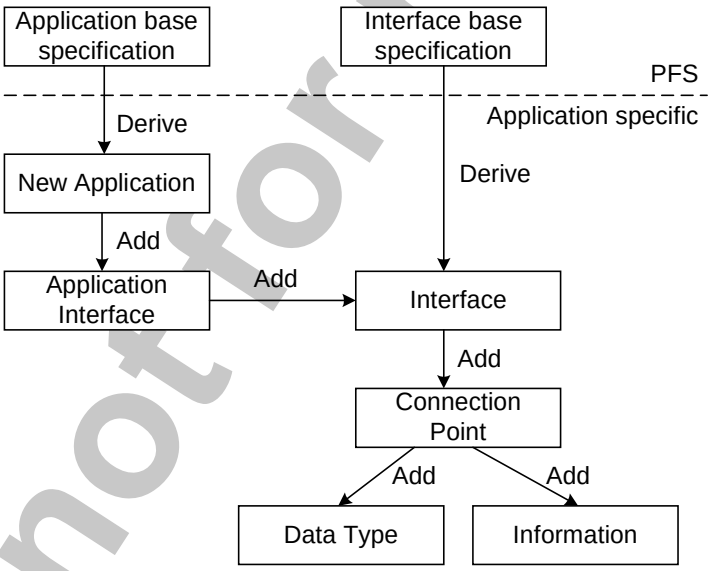

Figure 2 - Generating applications

Figure 2 shows the way to create an application specification derived from a template (“base specification”, i.e., the PFS).

As stated above, a new application specification must be created by deriving it from an existing PFS compliant specification. The newly created application will inherit all application interfaces from the respective base specification. The only mechanism allowed to adapt this newly created application specification to specific needs is to add application interfaces to it. A new application interface is created by grouping one or more existing or newly created interface specifications. As with application specifications, specifications of new interfaces have to be derived from existing specifications (i. e. abstract interface specifications of the PFS or other existing interface specifications). Newly created interfaces will inherit all connection points from the respective base specification.

Newly created interfaces can be adapted to specific needs by addition of connection points. Connection points already defined for that interface (i.e., those specified in base specifications of the interface in view) may not be changed. As shown in Figure 2, connection points form collections of data types or information specifications. They may contain zero or an arbitrary number of attributes (data types or information specifications). It is also allowed to have empty connection points. Data types and information specifications shall not be mixed in one connection point.

4.2 Products of the PCS

The PCS is basically a formalised way to describe the system interfaces of applications that use this standard. This means that the PCS must specify certain protocol entities that are used in the establishment of connections between applications and in the exchange of information. These entities are described in the following clauses.

There are also other entities produced from the PCSDL documents. These are either used in special protocol entities (priorities, load limitations) or in various PFS defined interfaces (authentication and application management). These entities are defined in the last two clauses.

Note that there is a difference in whether applications are clients or servers of a particular data object. These differences is based on the fact that the server will generally be less configurable than the client.

4.2.1 MAU name

The server MAU name is one of the connection attributes of the data objects used for communication. The *application* document defines the MAU name. Note that the document may specify that the MAU name shall be configurable, in which case the MAU name is defined during system integration by the help of some configuration tool.

4.2.2 Interface name

The interface name is common to a group of data objects that are connected to en block during connection establishment. The interface name is one of the data object attributes. The interface name is determined in the *application* document. The interface name can in some cases be configurable. In this case, the ability to configure the name will be made explicit in the *application* document.

4.2.3 Interface class name

If an interface name is changed during configuration, the old interface name as specified in the application document shall be saved in the application and made available to configuration tools as the “interface class name”.

The interface class name is not used during connection establishemnt and can in principle be disregarded by applications not conformant to the PFS.

4.2.4 Data object name

The data object name is another data object attribute that is used during connection establishment. The data object name is defined by the *interface* document. It is not possible to configure this name.

4.2.5 Data object function

The data object may represent one of several function types, e.g., read, write or subscribe. The functional capabilities of a data object is determined in the *interface* document. One data object can have only one function, but several data objects with the same name in the same interface

can be distinguished between by their function (or data structures). This is similar to name overloading in object oriented languages.

4.2.6 Data object structure

In addition to function, each data object is also recognised by the input and/or output data structures. The data structures are defined either in the *interface* document itself, in a *data type* document or in an *information* document. On the protocol level, it is only the structure of the data element that is important for establishing contact. However, the protocol has provisions for embedding information contents requirements in the data structure (see 4.2.7).

4.2.7 Data object information contents

For meaningful exchange of information, the participating applications need to know more about the sent and received data than just their structure. This aspect is partly covered in the *data type* document, where meaningful interpretations of data structures usually are defined together with the definition of the structure itself. It is also possible to give meaning to the data structures in the *interface* documents in conjunction with defining the functional scope of an interface. A third method is to require additional documentation from a provider of a server MAU. This requirement can be specified in the *application* document.

However, the preferred way to give meaning to data structures is through the *information* document type. This document can either be an addition to the data type document or replace the data type document all together.

4.2.8 Run-time library information items

Some parts of the companion standard documents generate attribute values for the run-time system that implements a MAU. These items are:

a) Load related attribute values, i.e., number of clients for an accept type interface or transaction queue length for all types.

b) Priority for various connection points or interfaces.

c) Password for interfaces.

d) Watchdog timer for the MAU.

These parameters are set in the application definition header and in conjunction with the definition of accept and connect interfaces in the same document.

Some of the attribute values are most commonly specified as configurable, e.g., password.

4.2.9 PFS information items

Parts of the companion standard documents do also generate various information items that are used by the PFS. These items i sused by software libraries and parts of the PFS to generate various configuration tables. These items are:

a) Version codes for PFS base class.

b) Authentication parameters in application and interface component documents for use in user authentication interface classes.

c) Manufacturer and model information for application management classes.

d) Original classes of interface classes that are renamed in application documents.

4.3 The PISCES Foundation Specifications (PFS)

The PFS contains the generic part of the PCS. The PFS consists of the following parts:

a) Application foundation specifications.

b) Definition of generic interfaces (see 4.4).

c) Data type specifications

Application foundation specifications form aggregations of standard interfaces according to the functionality to be covered by the respective generic application.

4.4 Generic Interfaces in the PFS

The purpose of generic interfaces specified in the PCS is to give the developer access to the services provided by the A-profile. Annexes in this fragment contains companion standards for the following interfaces:

a) Retrieval of general information about applications and interfaces like version codes, manufacturer and type of equipment.

b) Interfaces to change/assign control to a specific console combined with authentication of user and workstation, e.g., to change control, acknowledge alarms or change system parameters.

c) Specialised interfaces of a “system MAU“ associated with each LNA for system management on the application level, e. g. retrieve network statistics or report attributes of local MAUs.

d) General mechanisms for retrieving and manipulating data based on tagged information entities. These mechanisms include search, read, write, subscribe and alarm manipulation. The mechanisms can be used for general data access as well as for implementation of alarm systems. This includes an interface to transmit or receive stream based data

e) General interface for transmission of IEC 61162-1 telegrams over a system network.

5 The Companion Standard Reference Specification

5.1 Introduction

This clause contains the complete reference for the PISCES companion standard description language (PCSDL). It contains all information necessary to understand the companion standards contained in the PISCES foundation specifications (PFS) and to write own companion standards based on the PFS.

Clause 5.2 explains the basic concepts of the PCS. Clause 5.3 defines the general conventions with respect to the syntax and semantic of the PCS description language. It explains the use of tokens (identifiers, keywords and literal constants) and the use of white-space elements (delimiters, indentations, comments etc.). Clause 5.3.3 concludes with the explanation of naming rules with a particular focus on the scope rules for the identifiers.

As explained in 4.2, the PCS can be used to specify four types of entities: applications, interfaces, information and data types. These four different document types have a common structure, formed by a header specifying general properties for all definitions in the document and a body containing the individual definitions. The body of the definition document normally consists of several blocks, each identified by a keyword. This general structure will be elaborated on in 5.4.

Clauses 5.5 to 5.8 contain the specification of the description language for each documentation type mentioned above. Each of these sections have the following layout:

a) A short overview explaining the purpose and the properties of the document in question.

b) A description of the general layout of the respective document. This description includes the general properties than can be set within the header of the document and an overview of the blocks allowed in the body of the document.

c) A detailed reference to the syntax of the respective blocks.

5.2 Basic Concepts of the PCS

PCS supports a simplified object based principles for specifying protocol entities. To create new companion standards, the mechanisms of aggregation and specialisation can be used. Specialisation means that new companion standards can be derived from existing specifications. The new specification “inherits” all properties from the base specification it is derived from (see Figure 3).

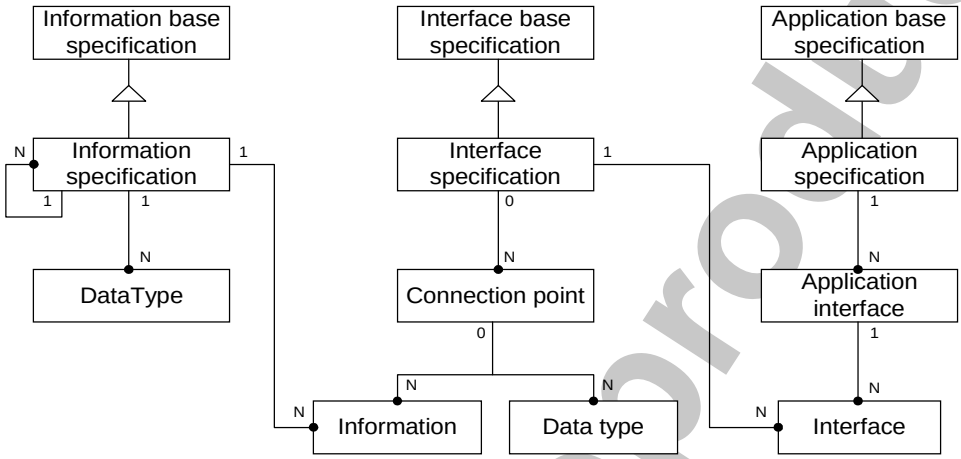

Figure 3 - Relationships between specifications of the PCS

The following types of specialisation are possible for PCS:

- Derived applications will inherit all application interfaces from the respective base specification.
- Derived interfaces will inherit all connection points from the respective base specification.
- Derived information specifications will inherit all attributes from their base specification. Attributes may be of the types data type or information.

Newly created specifications can be adapted to specific needs by adding properties to them. This mechanism is called aggregation. The following aggregation mechanisms are allowed for companion standards:

- Application specifications can be adapted by adding application interfaces to them. Each application interface consists of a set of interfaces. For every interface used, a corresponding interface specification must exist. Application interfaces already defined in base specifications may not be changed.
- Interfaces can be extended by addition of connection points. Connection points use one or more information or data types. For each information or data type used a corresponding specification must exist. Connection points already defined in base specifications can not be changed. Data types and information specifications shall not be mixed in one connection point.
- Information specifications can be adapted by adding new attributes to them (data types or information).

The mechanisms of specialisation and aggregation form relationships between the different specification types as elucidated in Figure 3.

5.3 Conventions for companion standard specification files

This clause covers general typographic conventions that apply to the PCS description files. It covers the use of tokens (5.3.1) and general naming rules including specification of the scope of named objects (5.3.2).

5.3.1 General principles

The files shall contain only characters from the 8 bit ISO 8859-1 character set, except where literal characters or strings are used. For literal character types, it is legal to use 16 bit characters where supported by tools and computer system.

NOTE – The actual representation of long characters must be checked in the programmer's references.

Only printable characters, the *newline* control code and *whitespace* are allowed.

No more than 80 printable characters (including *whitespace*) are allowed on one line. A continuation symbol can be used to continue a long line before a *newline* control code. The continuation symbol is the backslash, '\'.

5.3.2 Tokens

The basic element of the PCS description language is the token. A token in the PCS can be of one of the following types:

a) *Identifier*: Token to refer to a named entity in a specification. An identifier is defined by its name and scope.

b) *Keywords*: Reserved words of the PCS description language (see annex A).

c) *Literal constants*: tokens to express a constant value. Literal constants can be of several data types (e.g., integer, floating point, character or string).

d) Delimiters.

e) White space.

The following paragraphs give the typographic conventions that apply to these tokens.

5.3.2.1 Identifiers

Identifiers can be any sequence of letters or numbers. First character has to be a letter. Capitals and lower case letters will be distinguished between. All characters of an identifier are significant.

5.3.2.2 Keywords

Keywords are always in capitals. A list of the PCS keywords is given in Annex A. To enhance readability, keywords should be avoided when used out of context, e.g., in comments.

NOTE - Because the PCS description language distinguishes between capitals and lower case, it is generally legal to use keywords in lower case or in mixed-case variants e. g. for identifiers. However, to avoid confusion for the reader it is not recommended to do so.

5.3.2.3 Literal Constants

Literal constants are representing explicit values. Depending on the data type expressed by a literal constant, the following types can be distinguished:

a) *Integer constant*: representation of integers by a sequence of digits, e. g. `4123`. The number can be signed or unsigned, in hexadecimal, octal or decimal.

b) *Character constant*: representation of one character [ISO 8859-1], e. g. `'s'`. Long characters are also legal as literal where supported by computers and tools.

c) *Floating constant*: A floating constant consists of an integer part, a decimal sign, a fractional part, an optional **e** or **E** as exponential sign and an unsigned or signed integer exponent. The integer as well as the fractional part consists of a sequence of decimal digits. For instance `3.1415926` or `123.1e-5` are valid floating constants.

d) *String constant*: A string is a sequence of characters from the ISO 8859-1 character set, e. g. `"this is a string"`. Long characters are legal where supported by computer and tools.

e) *Aggregate constant*: This type is used to represent more complex data structures. Aggregates can be records or arrays of the type {<token1>, <token2>, ..., <tokenN>}.

A formal definition of the allowed formats of these literal constants can be found in the annexes of [IEC 61162-400].

5.3.2.4 Delimiters

Three types of delimiters are used:

a) *Whitespace* (see next clause) is used to separate other tokens in the documentation.

b) *Newline* is to be understood as the line separator character(s) used on a given platform, e. g. a single line-feed (on a UNIX system) or a carriage-return followed by a line-feed (on PC's).

c) *Block separators* (see section 5.4) are separated by two or more *newline* delimiters, i.e. with one or more empty lines between them.

5.3.2.5 White space

Whitespace is understood as one or more tokens that act as separators between tokens, but which have no syntactic meaning. Any contiguous number of the tokens listed below is defined to be "one" whitespace:

5.3.2.6 Indentation

For indentation the space character, ' ', shall be used. Indentations should be used to structure the text (e.g., mark one block).

5.3.2.7 Comments

Comments can be placed anywhere in the text to structure the document. The general operator that starts a comment to the end of a line is the semicolon ';'. Comments can be at the beginning of a line to mark the entire line as a comment or within a line – in the latter case the rest of the line is a comment.

NOTE - Newline following whitespace is a newline token even if the whitespace is part of a full line comment.

If a comment is extended over more than one line, it has to be marked with an asterisk, '*', as first character. A multiple line comment has to be terminated by a block delimiter (two or more newlines).

```
;----------------------------          Comment over an entire line.
DATA BLOCK Time
  * A representation of                Multiple line comment with
    relative time                      indentation, delimited by newlines.

word32_m   sec   ;seconds
word32_m   usec  ;micro-seconds        Declaration followed by comment
```

Figure 4 - Example comments

NOTE - Although the comments described here are not evaluated by a computer reading the description files, they have a significant impact on the human readability of PCS documents and are highly recommended.

5.3.3 Name structure

Identifiers for data types (interpretation, data blocks or constants), application, interface or information specifications may contain a mix of upper and lower case letters, digits and the special character underscore, '_'. They should normally start with an upper case letter.

NOTE - The A-profile uses these identifiers as attribute values during connection establishment, but does not enforce any rules on their construction other than that they do not contain the null character.

Other identifiers, e.g., attribute names, may contain the same mix of printable characters. For uniformeness in psecifications, they should start with a lower case letter.

5.3.4 Name scope

The scope of an identifier depends on its type. Identifiers declared within application, interface or information specifications are local to the respective specification. Dependent on specification type, this affects the identifiers as follows:

- *Applications*: Identifiers for interfaces specified within application specifications are local (5.5).
- *Interfaces*: Identifiers for connection points defined within an interface or identifiers for connection points defined in another interface referenced and reused by this interface is local to this interface (5.6).
- *Information specifications*: Identifiers for attributes to the information type are local (5.7).
- *Data types*: These are scoped dependent on being declared as local or global (see below).

Data type specifications contain two different sections, one for local and one for global variables. These sections are marked by the keywords `LOCAL` and `GLOBAL`:

- Data types (i.e., interpretations, constants and data blocks) within the `LOCAL` section are local to the respective specification. As for the specification types mentioned above, they are accessible from other specifications only via the scope operator, '.'.
- Data types within the `GLOBAL` section have global scope. This means that they are accessible from all specifications referencing the respective data type specification directly. In this case, the use of the scope operator is optional.

NOTE – The purpose of the GLOBAL keyword is to allow the definition of data types that can be used without scope operator in later documents. This simplifies the text of specifications.

The `LOCAL` section has to precede the `GLOBAL` part of data type specifications. If no keyword is used, the section is assumed to be global.

To make reference to a scoped identifier, the name of the scope it belongs to have to be prefixed and followed by a full stop, '.', delimiter. (e. g., `myInformation.m_myMember`).

5.3.5 Configurable identifiers and literal

Some identifiers, e.g., MAU and interface names may be specified as configurable during system installation. This is typically necessary if several identical units (MAUs) are installed in one system. In this case, each MAU must be given a unique name.

To specify in companion standard documents that an identifier or literal is configurable, it can be enclosed in pointed brackets, '<' and '>'. The documentation should specify where and how the entity is configured.

5.4 General Structure of PCS Specifications

5.4.1 General

PCS documents are used to specify the following entities:

a) Application specifications.
b) Interface (specifications).
c) Information specifications.
d) Data types.

An outline of the general document is shown in Figure 5.

The specification consists of two main parts:

a) The specification **header** defines attributes of a general nature and is similar for all document types.

b) The **body** of the specification consists of a sequence of **blocks** defining properties specific to the respective entity type. The structure of this part is dependent on the document type.

The header is delimited from the body by a block delimiter.

```
<type> <parameter_list>                  First line of specification indicates type.
  [* description]                        Comment to describe the entity specified.
                                         Each specification starts with a header setting
  <header keyword> <literal>             general attributes like version, date,
        . . .                            responsibility

;-----------------------------
;specification body
  <keyword> <parameter_list>             The body of a specification consists of a
    [* description]                      sequence of blocks giving the detailed
                                         specification of the entity in view.
    ;body of block

;-----------------------------
;next block
        . . .
```

Figure 5 - PCS general structure

5.4.2 The specification header

All specifications starts with a keyword defining the *type* of entity that shall be specified. `<type>` may be one of the following:

a) `APPLICATION` to specify an application

b) `INTERFACE` for an interface

c) `INFORMATION` to create an information specification

d) `DATA TYPES` to generate a data type specification.

A list of parameters is appended to `<type>`. The format of this list depends on the type specified. The first parameter of the list always specifies the identifier (i.e., the name) of the entity. For details on the syntax of the specific types see the following clauses.

The first line can be followed by a block comment giving a description of the content of the specification.

```
VERSION          <version_code>          One attribute definition
DATE             <date>                  Another after a newline,
RESPONSIBLE      <name>                  and another.
```

Figure 6 - Header

In the rest of the header, each line consists of a keyword naming an attribute to be specified and a literal constant giving the attribute value. An example is shown in Figure 6. The lines shall be separated by one or more newlines and/or block comments. The parser will normally look for the first keyword of the body of the specification to determine that the header has ended.

The following header fields are common and required for all PCS specification documents:

a) `VERSION <version_code>` The VERSION keyword sets the version of a PCS entity to the value given by <version_code>. The version code shall be written in the format N.N where N may be any integer-constant. Leading zeros are allowed (like e. g. 0.001).

b) `DATE <date>`: Defines the date of creation of the entity to `<date>`. The date shall be in the form YYYY-MM-DD with YYYY specifying the year of creation, MM the month and DD the day (all fields are integer-constants). For instance `2000-08-03` is a valid date.

c) RESPONSIBLE <name>: This field specifies the responsible organisation and author of the specification. It may be any string-constant. This field should normally name the organisation and person in the organisation responsible for further maintenance of the document.

5.4.3 The specification body

The basic element of the specification body is the block structure. Each block starts with a keyword indicating the type of block followed by a list of parameters. The available types of blocks are dependent on the type of entity to be specified. Details on legal blocks and the respective syntax will be given in the following sections. Blocks are separated from each other by at least one block separator.

5.5 Application specifications

5.5.1 Overview

An application specification is the description of the interfaces of one application unit that shall be connected to the IEC 61162-4 network. Each application specification consists of a set of one or more interface specifications. A PCS application specification, with its referenced documents, contains all necessary information to create the programming code that implements the interfaces of the application unit.

New application specifications can be derived from existing ones. The new specification inherits all properties from the existing application specification, i. e.:

- Attribute values defined in the header of the respective base specification
- All interfaces specified by the base specification

5.5.2 General Layout

Figure 7shows the general layout of the application specification. Examples of application specifications can be found in the annexes.

The keyword APPLICATION fefines the start of an application specification. The syntax of the statement is as follows:

APPLICATION <appl_name> DERIVED [FROM] <base_application>

The identifier <appl_name> defines the name of the application (the MAU name). The keyword DERIVED shows that the application is derived from a base specification given by the identifier <base_application>. This identifier has to be a valid reference to another application specification. The keyword FROM is used to make the statement more readable. A tangible application specification shall be derived from a base specification. Only abstract specifications (specifications part of PFS) need not to be derived from base specifications.

```
APPLICATION <appl_name> DERIVED [FROM]
<base_application>

  <general header>


;-----------------------------------------------------
-
 REFERENCES
  <interface_name>  [version]
  . . .

;-----------------------------------------------------
-
 USAGE
  [* description]

;-----------------------------------------------------
--
INTERFACES
  [* description]

  ACCEPT <interface_name>
    [* description]

   <properties of the interface>
       . . .

  CONNECT <interface_name> ON <server_name>
    [* description]

    <properties of the interface>
    . . .
```

Figure 7 - General layout of an application specification

A block comment should follow the first line, giving a short overview of the application specified. It is recommended also to supply also a revision history where appropriate. This is not part of the formal syntax and can be included after the description by another comment block.

5.5.3 Header

As for all entity types covered by the PCS, an application specification is divided into a header and a body. In addition to the general properties listed in clause 5.4 the following entries can be defined in the header of application specifications:

```
VERSION          <version_code>
DATE             <date>
RESPONSIBLE      <name>
[MANUFACTURER    <name>]
[MODEL           <name>]
[WATCHDOG        <interval ms>]
[AUTHENTICATION  [<user>:<password>,< user>:<password>, . . . ]]
```

Figure 8 - Application header

a) `MANUFACTURER <name>:` this attribute specifies the manufacturer of the application specified here. Name may be any character string.

b) `MODEL <name>:` the model name of the application should be given here. This attribute is used e. g. to distinguish between different variants of a product family. Name may be any character string.

c) `WATCHDOG <interval ms>:` this field specifies an optional watchdog interval (in milliseconds) that is used by the LNA to periodically interrogate the state of the MAU.

d) `AUTHENTICATION <user>: <password>` This field specifies that authentication is used for all interfaces in the MAU and may, optionally, specify available user and password codes. The latter is normally not included in open documents. Authentication uses a dedicated PFS interface.

5.5.4 Body

The specification body shall consist of three main blocks in the order shown in the following clauses. All blocks shall be present, although they may be empty or consist of just comments as, e.g., the USAGE block.

5.5.4.1 References

Each external interface has to be referenced before it can be used in the INTERFACES blocks. This is done in a block that starts with the keyword REFERENCES. Each additional line of this block contains a reference to an interface in the form:

```
<interface_name>  [version]
```

<interface_name> shall be the name of a valid interface. To distinguish between different versions of one interface the version code may be specified as a text string. This string, if present, will be compared to the version code specified in the header of the referenced interface. A mismatch shall cause a parser error.

5.5.4.2 Usage

This block gives a description on how the application should be used. The block starts with the keyword USAGE, but the remainder of the block must be formatted as a block of comments.

5.5.4.3 Interfaces

The INTERFACES block contains the specification of the application interfaces the application is provided with. Each application interface identifier is prefixed by the keyword ACCEPT (for a server interface) or CONNECT (for a client interface). The keyword line may be followed by a block comment giving a description of the purpose and functionality of the respective application interface.

5.5.4.4 Accept type interface

The syntax used to specify a server interface is illustrated in Figure 9.

```
ACCEPT <interface_name>
  [* description]

  [MAX MESSAGE RATE  <msg_per_sec>]
  [AUTHENTICATION    [<user>:<password> . . .]]
  [CLIENTS           <numb_clients>]
  [TRANSACTION QUEUE <num_trans>]
  [PASSWORD          [<passwd>]]

  INTERFACE [COMPONENT] <name>   . . .
```

Figure 9 - Accept interface template

The specification of a server interface starts with the keyword ACCEPT. The newly created interface will be named <interface_name>. The name may be specified as configurable by enclosing it in pointed brackets. The interface will be composed of a number of interface components as specified at the end of the example.

A number of properties can be specified for the server interface by using a set of property definition statements. These statements must be placed between the header line and the first interface component definition. They may be listed in any order. Each statement shall be one line, separated from other lines by one or more newlines. The following properties may be set for a server interface:

a) MAX MESSAGE RATE <msg_per_sec>: This entry is not converted to a protocol entity. It is used to check and verify system and module load. It shall specify the maximum number of transactions (including connection attempts) that the server application accepts on this interface.

b) AUTHENTICATION <user>:<password>: If user authentication is required for the respective interface, this property has to be defined. The user and password fields may optionally be specified to list the available users. The password may be specified as configurable.

c) CLIENTS <numb_clients>: the maximum number of clients that are allowed to be connected to the server may be specified here. The token <numb_clients> is of integer type. This property may be omitted if the maximum number of clients is not limited. The limitation is enforced by the LNA by denying further connection attempt, after the number of clients has been accepted by the MAU. The limitations do not apply to *urgent* connection attempts.

d) TRANSACTION QUEUE <numb_trans>: the maximum number of pending transaction requests that the server allows in the input queue may be specified here. The token <numb_trans> is of integer type. This property will limit the number of transactions sent to the server for a given interface. By delaying acknowledgements, the server can use this limit to control its load. The limitation does not apply to *urgent* requests.

e) PASSWORD [<passwd>] If the server shall be protected by a password it may be specified here. Note that the password normally will be configurable and that the string specified as <passwd> usually is empty or a string in pointed brackets. The string will be the default password for the interface.

The interface connection points are specified by a number of the following statements:

```
INTERFACE [COMPONENT] <name>
```

Each application interface consists of one or more interface components, each specified as in the line above. All components specifications must be in one block (only one newline between each line). The keyword [COMPONENT] is normally used to make the code more readable. <name> has to be the name of a valid interface specification as defined in the REFERENCES section (see above). The complete interface (consisting of all components) will be instantiated as one A-profile interface and given the name set in the ACCEPT statement.

5.5.4.5 Connect type interface

The specification of a client interface is initiated by the keyword CONNECT. The definition is similar to the accept type interface definitions, except for the prperties that can be set and the definition of a server to connect to.

The identifier <server_name> has to be a valid name of an application acting as server for the respective application interface (the server MAU name). The <interface name> must likewise be the name of one of the application interfaces on the server. Both these names may be specified as configurable.

```
CONNECT <interface_name> ON <server_name>
  [* description]
  [MAX MESSAGE RATE  <msg_per_sec>]
  [PRIORITY          [interface_comp[.conn_point]]  <priority>]
  [TRANSACTION QUEUE <num_trans>]
  [PASSWORD          <passwd>]
    . . .
  INTERFACE [COMPONENT] <name>
    . . .
```

Figure 10 - Connect application interface template

The following properties may be set for a client interface:

a) MAX MESSAGE RATE <msg. per sec.> This entry has the same function as for the ACCEPT statement. It shall specify the maximum number of transactions generated by this client application for this interface. This is a measure that may be enforced by the application, but it is normally only tested against and is used as guideline for total system load calculations.

b) PRIORITY [interface_comp[.conn_point]] <priority> The PRIORITY flag can be used to set the priority of an interface component or a specific connection point. If not

specified, the priority will be set to NORMAL. The connection point of the interface component affected by this property is specified by the name argument. If the name argument is omitted, the priority applies to all connection points in the application interface. If the .conn_point component is omitted, the priority applies to all connection points in the interface component specified. The constant-token <priority> shall have the values NORMAL or URGENT.

c) TRANSACTION QUEUE. This is as for the accept interface.

d) PASSWORD. This is as for the accept interface, except that the password specified is that used for connection to the server.

The remainder of the specifiction consists of one or more lines as follows:

```
INTERFACE [COMPONENT] <name>
```

These lines are used to specify the components of the application interface in the same format as for the accept interface.

Note that a connect interface may contain a sub-set of the components that the server supports in its interface. It cannot, however, have components that is not supported by the server. This will result in a run time connection failure.

5.5.4.6 Handling of anonymous broadcast

If the application interface name is ABCMn or ABCn where n is a digit from 1 to 5, the interface components shall consist of only anonymous broadcast connection points. For accept interfaces, this means that a MAU name is disregarded.

These connection points shall be exported as listening (connect) or sending (accept) on the specified broadcast port.

5.6 Interface component specifications

5.6.1 Overview

Interfaces are used in application specifications to describe the functionality and connectivity of the application. The application specification defines application interfaces which consists of a number of interface components. Each of the interface components is defined in an interface definition document.

The format of interface component descriptions is defined in this clause. Interface components themselves form aggregations of connection points. Each connection point specifies input and/or output information available from the interface.

NOTE - In essence, interfaces describe the view on an application from the “outside world” – from an object oriented point of view, interfaces specify the methods giving access to the internal structure and behaviour of an application object. That means that all functionality and behaviour of an application visible from other applications is given by the interfaces.

5.6.2 General Layout

Figure 11shows the general layout of the specification for an interface component. Examples of interface specifications can be found in the annexes. The following clauses describe the document in more detail.

```
INTERFACE <interface_name> DERIVED [FROM] <base_specification>

  <header>

;-------------------------------------------------------
 REFERENCES
  <info_name> [version]
  <data_type> [version]
       . . .
;-------------------------------------------------------
 USAGE
  [*description]

;-------------------------------------------------------
REQUIRED DOCUMENTATION
  <entity_name>   [version]
  . . .

;-------------------------------------------------------
CONNECTION POINTS
  <connection_type> <connection_name>
    [* description]

    INPUT
      [* description]

      <element declaration>
      . . .

    OUTPUT
      [* description]

      <element declaration>
      . . .
```

Figure 11 - General layout of an interface specification

5.6.3 Header specification

The keyword `INTERFACE` defines the beginning of an interface component specification. The syntax is as follows:

```
INTERFACE <interface_name> DERIVED [FROM] <base_specification>
```

The identifier `<interface_name>` defines the name of the interface specification. This is later used in application specification documents to identify the interface component. A block comment should normally follow the first line, explaining properties and usage of the interface component. The keyword `DERIVED` indicates that the interface is derived from a base specification given by the identifier `<base_specification>`. This identifier has to be a valid reference to another interface specification. The keyword `FROM` can be used to make the semantics easier to understand.

It is mandatory to derive a tangible interface specification from a base specification. Only abstract specifications (parts of the basic PFS) need not to be derived from base specifications.

As for all entity types covered by the PCS, an interface specification is divided into a header and a body. The properties to be set in the header are given in 5.4.

5.6.4 Body Specification

The blocks defined in the following clauses shall be part of the body of an interface specifications.

5.6.4.1 References

Information or data type specifications used in the specification of connection points must be referenced before use. Each line of the `REFERENCES` block contains a reference to an information or data type specification in the form:

```
<name> [version]
```

<name> shall be the name of an information or data type specification. To distinguish between different versions of specifications referenced here the version code may be given by the token <version>. This is a string that, if specified, must match the version string in the referenced document.

5.6.4.2 Usage

This block shall give a description on how the interface should be used. The actual contents is a comment and is not parsed.

5.6.4.3 Required documentation

This block lists additional documentation required for proper interpretation of the interface specification. This is mainly to instruct the parser/compiler of the PCS specification that it should give a warning to the system integrator to check the availability of the specified documentation. It will not generate protocol entities. Following the keyword a list is given as one text block; each line in the block should normally contain an entry of the form

```
<entity_name>   [version]
```

The token <entity_name> is a user defined symbol which one can assume identifies the documentation in question. If version information is available for the documentation, it may be specified by the token [version].

5.6.4.4 Connection points

This block contains the specification of the connection points the interface component provides. Each connection point is specified by the following parameters:

a) <connection_type>: Keyword specifying the type of the connection as in the first column of Table 1.

b) <connection_name>: Identifier specifying the name of the connection point to be created.

c) INPUT/OUTPUT: Dependent on <connection_type> (see Table 1) the connection point needs an INPUT and/or an OUTPUT specification. The formats of both input and output blocks are the same.

The connection types and corresponding input and output requirements are listed in the below table. Note that an input or output field may be empty although the existence of the field is required.

Table 1 - Connection point types

Keyword	In	Out
FUNCTION	Yes	Yes
READ	Yes	
WRITE		Yes
NONACKED WRITE		Yes
SUBSCRIBE		Yes
INDIVIDUAL SUBSCRIBE	Yes	Yes
BROADCAST SUBSCRIBE		Yes
ANONYMOUS BROADCAST		Yes

The <element declaration> can take one of the following forms:

a) <information> <name>, the specification of a single information entity (previously declared in the references section). This is then the output or input entity.

b) `<data type> <name> [. . .]`, the specification of a data type with the syntax as for a data block (5.8.4.4.3). The single type, the array or the data block will then be the input or output entity.

NOTE 1 – If a new data block is defined (over multiple lines) in this manner it may be given a name by the parser for reference by application source code. The name will normally be the name of the connection point with the postfix `In` or `Out`. This is, however, determined by the parser.

NOTE 2 – The use of a single data item or array will not cause a new data block to be defined. In this case, only the actual data item or array is transmitted.

c) `opaque <information> [count type:max size]`, the specification of a variable length data block (as for variable length arrays in 5.8.4.4.3), where the interpretation is defined in a n information document previously referenced.

The `<name>` field may be used by the parser to name application source code entities. It has no meaning for any A-profile entities.

5.7 Information Specifications

5.7.1 Overview

Information specifications define information transmitted via a PISCES network. Mainly, two properties of information entities will be defined:

a) Interpretation: The object based concept (i. e., assignment of information entities to a specific type) and the transmission of additional information by complex data structures allows the programmer to transmit data with an implicit “meaning”, see 4.1.

b) Structure: For high-level applications (e. g. decision support systems) it is not sufficient to process only data. Additional information has to be available, e.g., source of information, accuracy and time stamp. This leads to complex structures. The specification of the internal structure of such complex data entities can be given in information specifications.

5.7.2 General Layout

Figure 12 shows the general layout of an information specification. Examples can be found in the annexes.

```
INFORMATION <information_name> [DERIVED [FROM] <base_specification>]

  <header>

;-----------------------------------------------------
 REFERENCES
  <type_name> [version]
            . . .

;-----------------------------------------------------
 USAGE
  <description>

;-----------------------------------------------------
ATTRIBUTES
  [* description]
  <attribute_list>
```

Figure 12 - General layout of an information specification

5.7.3 Header

The keyword `INFORMATION` defines the beginning of an information specification. The syntax is as follows:

```
INFORMATION <information_name> [DERIVED [FROM] <base_specification>]
```

The identifier `<information_name>` defines the name of the information specification. This name can be used by interface component specifications to specify an input or output to a connection point. The optional keywords `DERIVED FROM` indicates that the information specification is derived from a base specification given by the identifier

<base_specification>. This identifier has to be a valid reference to another information specification. The keyword FROM is used to make the text easier to read. A block comment should follow the first line, explaining properties and usage of the information entity.

As for all entity types covered by the PCS, an information specification is divided into a header and a body. The properties to be defined in the header are defined in 5.4.

5.7.4 Body

The body consists of the blocks defined in the following clauses in the order they are described.

5.7.4.1 References

The types used within the ATTRIBUTES block described below have to be referenced before usage. Data types (5.8) and other information specifications can be referenced here. Each reference has to be in the form:

```
<type_name> [version]
```

<type_name> has to be a valid name of an information specification, data block or interpretation. To distinguish between different versions of one type the relevant version number may be specified by substituting [version] with the relevant version string.

5.7.4.2 Usage

This block gives a description on how the information specification should be used. The field <description> is a comment block and is not used by any parser.

5.7.4.3 Attributes

This block specifies the attributes of the information entity. Each attribute is declared on a separate line, making up the block <attribute_list>. Each line use the following syntax:

```
<type_name> <attribute_name>
```

Where <type_name> may be one of the following:

a) A valid name of another information entity referred to in the REFERENCES block (see above)

b) A valid name of an interpretation or data block specified in a referenced data type document (5.8).

c) One of the basic data types (see annex B).

An <attribute_name> is specified so that the parser can generate application code for the information element.

5.8 Data Types

5.8.1 Overview

Data type specifications have three purposes:

a) Define new data types. A data block defines a new data type. This type is normally also associated with an implicit interpretation.

b) Define interpretations**.** An interpretation describes how to *understand* the contents of data of a given type. The interpretation declaration gives an old type a new name and defines a new interpretation for this new type.

c) Define named constants. The constant declarations define constants with symbolic names that can be used by the same or other PCS documents.

The body of a data type specification is divided into two sections. The section identified by the keyword LOCAL contains all data types having local scope. Data types with global scope are included in the section marked with the keyword GLOBAL, see 5.3.3. Unless a data type, interpretation, or constant is declared GLOBAL, the respective entity can only be accessed from other specifications by using a scope operator. This is done by using a concatenation of the data type specification name and the data type name with a dot (‘.’) as separator.

5.8.2 General Layout

Figure 13 shows the general layout of the specification of data types. Examples can be found in the annexes.

```
DATA TYPES <module>

  <header>

;-------------------------------------------------------
 REFERENCES
  <module> [version]
   . . .

;-------------------------------------------------------
LOCAL
  * specify data types with local scope

CONSTANT <type_name> [OF <old_type>] IS <constant-token>
  [* description]

;-------------------------------------------------------
INTERPRETATION <type_name> OF <old_type>
  [* description]

  <interpretation>
       . . .

;-------------------------------------------------------
DATA BLOCK <type_name>
  [* description]

  <data_list>
       . . .

;-------------------------------------------------------
UNION <type_name>
  [* description]

  <data_type>  <item>  [;description]
  <data_type>  <item>  [;description]
  <data_type>  <item>  [;description]
       . . .

;-------------------------------------------------------
GLOBAL

  * specification of data types with global scope, syntax is as
    in the LOCAL section of the specification
```

Figure 13 - General layout of a data type specification

5.8.3 Header

The keyword DATA TYPES defines the beginning of the data type specification. The complete syntax is as follows:

```
DATA TYPES <module>
```

The identifier <module> gives the name of the data type specification. A comment should follow the first line, explaining properties and usage of the data types specified. Other header fields are described in 5.4.

5.8.4 Body

The body of a data type specification consists of the following blocks in the listed order.

5.8.4.1 References

External data type specifications necessary to define the entities of the current specification shall be referenced here. This block consists of a list with each line containing a valid name of a PCS data type specification. To distinguish between different versions the version concerned here may be given by the token <version>.

5.8.4.2 Local data definitions

The keyword LOCAL introduces the specification of data types with local scope. These data types are accessible from other specifications only via the scope operator '.', see 5.3.2. The section marked as LOCAL has to precede a GLOBAL section, if the LOCAL section is used.

5.8.4.3 Global data definitions

The keyword GLOBAL precedes the specification of data types with global scope. These data types are accessible from other specifications directly without the use of the scope operator ('.' – see 5.3.2). The LOCAL section must precede the GLOBAL section if a LOCAL section is used.

5.8.4.4 Data definition types

Each local or global block consist of a number of data type definitions. These definitions can be in any order. The following clauses specify the format of each type of definition.

5.8.4.4.1 Constant

The format of a constant definition is:

```
CONSTANT <type_name> [OF <old_type>] IS <constant-token>
```

The new constant named <type_name>, optionally specified to be of type <old_type>**,** is given a constant value <constant-token>**.** <constant-token> can be any literal representable by the optionally specified type, see 5.3.1. The type has to be a built-in type or a type that has been defined earlier or in a referenced document.

5.8.4.4.2 Interpretation

The format of an interpretation is:

```
INTERPRETATION <type_name> OF <old_type>

     <interpretation>
        . . .
```

This block defines a new data type named <type_name> based on some built-in or previously defined data type named <old_type>. The defined interpretations follows after a block separator. Each line in the interpretation block defines a meaning for one discrete value that the type is able to represent. The different variants are shown in the below table.

Table 2 - Interpretation forms

Form	Description
literal = token	Define a new token to have a constant value (literal)
token2 = token1	Define a new token1 to be identical to an old token2.
token	Define a new token with no particular value
constant	Describe the interpretation of a specific value

All forms can be supplied with an in-line description after a semicolon. The two variants using an equals signs assign a new value to the left hand side token. The new token can be used in other contexts in the definition documents. The constant tokens can be of one of the types described in 5.3.1 or a symbol created in a constant definition.

The two last lines in the table are used to give informal description of how a particular value or a previously defined token or constant should be interpreted. Neither of these will define new symbols.

5.8.4.4.3 Data block

The data block is used to define a new data type consisting of a set of data elements (a record). The structure of the data block definition is:

```
DATA BLOCK <type-name>

  <data-type> <element-name> [; description]
  <data-type> <element-name> [; description]
     . . .
```

The new record is given the name `<type-name>`. After the keyword line a comment should follow with an informal description of the new type.

Each data element in the new record is listed in order after the description. Each line defines one new element of type `<data-type>` with name `<element-name>`. The different elements and the declaration forms are shown in the below table. `old-type` references a previously defined data type or a built-in type as listed in annex B.

Table 3 - Data type declaration formats

Form	Description
old_type	A single element of given type
[N] old_type	An ordinary array of given type
[wtype:N] old_type	Variable length array of given type

The N can be any integer. It specifies the (maximum) number of elements in the array. The third form defines a variable length array. `wtype` may be one of the unsigned integer types listed in annex B.

The built-in types are listed in annex B. These types should be referenced with the name given in the left hand column.

5.8.4.4.4 Union

Unions are data objects that can be transmitted as one of several data elements of different types and sizes. The specification of a union contains the possible elements that can be transmitted in the place of this object The syntax of a union specification is shown below:

```
UNION <type-name> : <wtype>

  <data_type> <item> [; description]
  <data_type> <item> [; description]
     . . .
```

The union is given the name `<type-name>`. The *wtype* is an unsigned integral basic type that is used to transmit the enumeration value of the union element actually transmitted. The enumeration starts at one for the first line and increasing continuously. The value zero is reserved for an empty union, i.e., no data transmitted.

Only one of the elements defined on the following lines will be transmitted. Each element must be a basic type or a derived type (data block or interpretation).

6 PISCES Foundation Specification (PFS)

6.1 Introduction

This clause gives an overview of the components of the PFS and defines the structure and naming conventions.

6.2 Naming conventions

To simplify the reading and understanding of PFS class trees, the following naming convention has been defined:

a) All PFS class names start with three upper case letters.

b) PAC is used for application classes.

c) PCC is used for interface component classes.

d) PIC is used for information classes.

In addition, the PFS will contain one standard data type file called "`General`". Other data type files are created as found necessary.

6.3 Application classes

6.3.1 Introduction

The PFS application classes are organised in a tree as shown in Figure 14. Each box represents one application base class. The shaded boxes represents applications that are not included in the companion standard source code, but which are included as examples of possible derivations. The name of the application class is in bold font in the first line of text. The component interfaces are listed underneath.

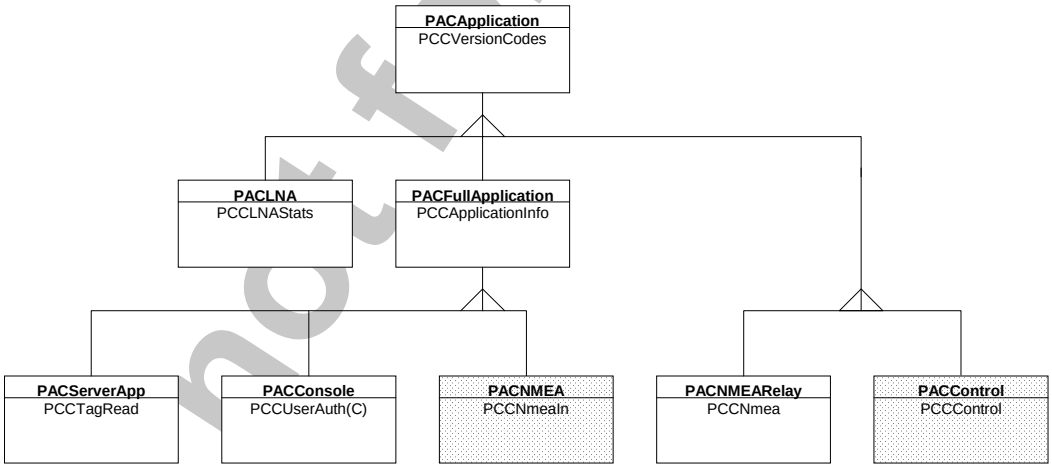

Figure 14 - Application class tree

New tangible applications can be derived from any class in the tree, also those that are not leaf nodes.

The tree defines a set of generally useful application classes as described in the following clauses.

6.3.2 Application base class: PACApplication

All PFS compliant applications must as a minimum have version and manufacturer codes available (`PCCVersionCodes`). These components of the PFS are described in Annex C (in the PCSDL format).

Two “simple” example applications are shown: A simple relay function for IEC 61162-1 telegrams (PACNMEARelay) and a simple control function (PACControl). The latter is assumed to be some kind of simple control interface.

It is recommended that only very simple applications are derived from this base class. Most applications should be derived from the managed application base class.

6.3.3 LNA MAU Application: PACLNA

There is an application class associated with the system’s LNA (PACLNA). This application is implemented as a MAU that can look into some of the LNA’s management structures. This can be ussed for application management, configuration and debugging. The minimal functionality of this MAU is defined in Annex C.

6.3.4 Managed applications: PACFullApplication

There is also a more complex class of applications that as default support the inspection and possible configuration of the interfaces and connection points implemented by the application (PACFullApplication). The source code for the interfaces is included in Annex C.

This application class enables, together with the LNA MAU, the on-line creation of a connectivity tree with detailed information about each connection point. This is an important tool for system integration and system amnagement and debugging.

6.3.5 NMEA Application: PACNMEARelay

One example of a full application is the NMEA relay base class. This base class allows the relay of IEC 61162-1 telegrams over the IEC 61162-4 protocol. This represents a simple way to integrate parts 1, 2 and 4 of this standard. The source code is included in Annex E.

6.3.6 Console application: PACConsole

The console application base class provides a starting point for the cretaion of consoles that shall cater to user centered HMI. In this standard, the base class is only provided with an authentication client interface (PCCUserAuth). This is typically used to verify console position and user authentity before any operation with side effects are allowed to be performed. The source code is included in Annex C.

6.3.7 General Alarm and Monitoring Application: PACServerApp

A simple base class with a simple data base function for general data retrieval is implemented in this application base class. A more detailed application tree is described in Annex D. This base class, or at least parts of its component interfaces should be included in any application that may present data to a higher level application.

These application classes are also useful as general gateways between any other sub-system or bus and the IEC 61162-4 protocol.

7 Specification requirements for PCS compliant applications

7.1 Introduction and general documentation format

To enable the user to get a reasonable overview of an application’s functionality and the relationship to and between interfaces, a set of documentation requirements has been developed.

The application documentation shall consist of three main parts as described in the following clauses.

7.2 Function block

7.2.1 Function block graphical view

The documentation shall contain a graphical function block view. This view shall be structured as indicated in Figure 15. Other graphical formats can be used, but the information entities listed in the following shall as a minimum be included.

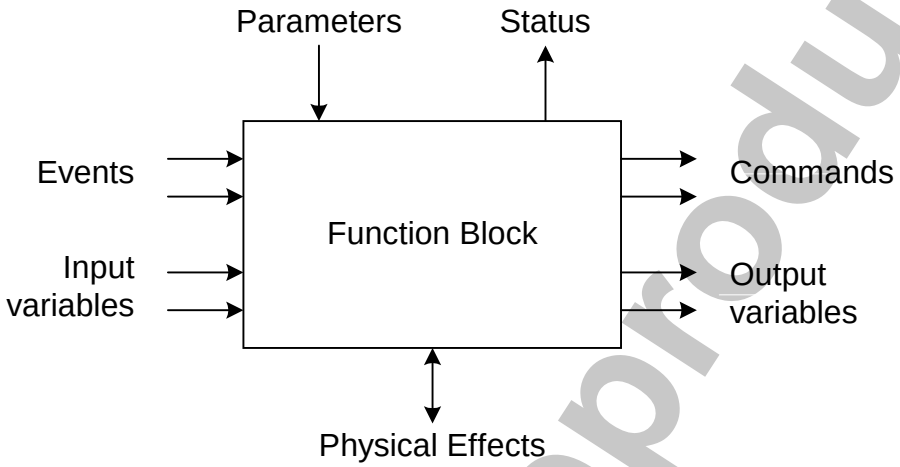

Figure 15 - Function block prototype

In the function block view, the application is drawn as a rectangle with certain inputs and outputs. The inputs and outputs should be labelled with the corresponding connection point, interface component or application interface dependent on wanted resolution. Only the inputs and outputs that are most important for application function need to be included. The figure shows those inputs and outputs that are normally included. These are discussed in the following clauses.

7.2.2 Physical effects

Physical effects are interactions not representable in the IEC 61162-4 network. They shall be drawn as double pointed arrows underneath the function block.

Physical effects are effects of interaction between the physical entities outside the control system and the function block, transformed by e.g. sensors, actuators or HMI.

7.2.3 Input variables

Input variables shall be drawn as arrows pointing into the function block in the lower left corner.

These are information elements generated by other function blocks. They are read during the execution of the function block's algorithm and are necessary for the correct operation of the function block.

7.2.4 Output variables

Output variables shall be drawn as arrows pointing out of the function block in the lower right corner.

These are readable information elements generated as a result of execution of the function block. These are typically input to other function blocks.

7.2.5 Events

Events shall be drawn as arrows pointing into the function block in the upper left corner.

Events are commands or notifications that are delivered to the function block from other function blocks and which cause the function block to perform some operation.

7.2.6 Commands

Commands shall be drawn as arrows pointing out of the function block in the upper right corner.

Commands are generated by the function block as commands or notifications (including alarms and warnings) to other function blocks to trigger some remote operation.

7.2.7 Status

Status shall be drawn as arrows pointing up from the function block in the upper right corner.

Status represents information items that can be made available to other function blocks that are not yet known. This is typically interfaces to various forms of higher level decision support systems.

7.2.8 Parameters

Parameters shall be drawn as arrows pointing down into the function block in the upper left corner.

Parameters represents data that can be set in the function blocks to control the function block's operation. The operation of the function block shall not depdent on the continous availability of these data. Typically this can be filter constants, set-points or alarm limits.

7.2.9 Indication of accept or connect functionality

It may be convenient to indicate in the figure what is defined as connect (client) and what is defined as accept (server) type interfaces. Note that this is independent of an entry point acting as input or output. The following conventions shall be observed:

a) A client entry point shall be labelled with an upper case 'C' near the rectangle.

b) A server entry point shall be labelled with an upper case 'A' near the rectangle.

c) No label is legal and does not carry any particular meaning.

7.3 Functional description

The documentation shall contain a section that describes the functionality of the application. This section shall relate the various inputs and outputs to the overall functionality.

The level of detail and the actual mechanisms employed to describe the functionality is left to the author. As a general rule, the description should allow a reader to understand what the application does and how the various inputs and outputs can be used to deploy the application in an integrated system.

7.4 Companion standard descriptions

The third part of the documentation is a detailed description of all inputs and outputs in the form of PCSDL documents. Base classes that are used in derivations and which are taken from official versions of this standard, need not be included in the listings.

Interfaces that are only used for internal purposes and which are not meant for public use may also be deleted from the listing providing that these interfaces have no safety related impacts on the application or in the system the application shall operate in.

Annex A
(Normative)

Defined keywords

The following table shows the keywords reserved in the PISCES companion standard definition documents.

Keyword	Description
ACCEPT	Define an accept-type interface
ANONYMOUS BROADCAST	Define an ABC connection point
APPLICATION	Define a new application specification
ATTRIBUTES	Declare attributes of an information specification
AUTHENTICATION	Specify need for user authentication
BROADCAST SUBSCRIBE	A data object function type
CLIENTS	Specify maximum number of clients for an interface
COMPONENT	Syntactic sugar
CONNECT	Define a connect-type interface
CONNECTION POINTS	Define a set of connection points
CONSTANT	Define a constant data item
DATA BLOCK	Define a new data aggregate type
DATA TYPES	Define new data types
DATE	Followed by date of last modification
DERIVED	Specify the base specification of a PCS specification
FROM	Syntactic sugar
FUNCTION	A data object function type
GLOBAL	Define global scope for an identifier
INFORMATION	Define a new information specification
INDIVIDUAL SUBSCRIBE	Define an individual subscribe connection point
INPUT	Define the data object's input record
INTERFACE	Define a new interface specification
INTERFACES	Define application interfaces
INTERPRETATION	Define an interpretation of a data item
IS	Syntactic sugar
LOCAL	Define local scope for the following data types
MANUFACTURER	Specify manufacturer of an application
MAX MESSAGE RATE	Specify maximum message rate of an interface
MODEL	Specify model of an application
NAMED	Syntactic sugar
NONACKED WRITE	A data object function type
OF	Syntactic sugar
ON	Syntactic sugar
OUTPUT	Define the data object's output record
PASSWORD	Specify password protection for a server interface
PRIORITY	Specify the priority of a CONNECT interface
READ	A data object function type

Keyword	Description
REFERENCES	Heads “included files“ section
REQUIRED DOCUMENTATION	Additional required documentation for an interface
RESPONSIBLE	Responsible author of a specification
SUBSCRIBE	A data object function type
TRANSACTION QUEUE	Specification of load limitation for server
UNION	Specify a union data type
USAGE	Heads a description of the usage of a specification
VERSION	Followed by version code of the specification
WRITE	A data object function type

Annex B
(Normative)

Basic PISCES Data Types

The following basic data types are supported by PISCES. The PISCES protocol guarantees to transmit any aggregates of these types between application units with no loss in precision or value.

Type	Description
bool_m	1 bit Boolean
char8_m	8 bit character
char16_m	16 bit character
word8_m	8 bit unsigned integer
wordl6_m	16 bit unsigned integer
word32_m	32 bit unsigned integer
word64_m	64 bit unsigned integer
int8_m	8 bit 2's complement integer
intl6_m	16 bit 2's complement integer
int32_m	32 bit 2's complement integer
int64_m	64 bit 2's complement integer
float32_m	32 bit floating point
float64_m	64 bit floating point

Refer to a more detailed description and a formal definition of representation in [IEC 61162-401].

Annex C
(Normative)
General applications companion standards

C.1 Introduction and general principles

This annex describes interface components and applications for basic system management functions. Each application shall belong to one of the super-classes:

a) `PACSimpleApplication`: Shall provide mechanisms for accessing simple information about the application (version codes, manufacturer and model information).

b) `PACFullApplication`: Shall provide mechanisms for accessing information about the interfaces and connection points provided by the application.

c) `PACLNA`: Special application associated with LNA.

Other applications are defined to be not conformant with the standard.

This annex also contains an authentication interface that can be used by applications that require operator or workstation authentication.

C.2 Functionality overview

C.2.1 General data definitions

The data type `General` provides general data definitions for all applications conformant to this standard.

C.2.2 Version codes

The interface `PCCSimpleApplication` provides version codes in standard three number format. Version codes for application program and protocol shall be provided.

C.2.3 Manufacturer and model identification

These are text strings that identify the manufacturer and the model. They should be the official name of the respective entities.

C.2.4 Interface and MCP information.

The `PCCFullApplication` interface provides additional information that allows a management client to inspect all interfaces and MCPs.

C.2.5 Authentication

Authentication is application specific in that different servers have different requirements for authentication. The standard provides a general interface to do authentication and this, together with the session codes, can be used to make sure that the identity of a requesting client is that which one expects.

C.2.6 File overview

The following table lists the interfaces and applications are contained in this annex.

Table 4 - Application related CS components

Component	Description	File
General	General data types	general.mcs
PACSimpleApplication	Simple application skeleton	appsimp.mcs
PACFullApplication	Full application skeleton	appfull.mcs
PCCVersionCodes	Interface for version codes	version.mcs
PCCApplicationInfo	Interface for application information	appinfo.mcs
UserAuth	User authentication data types	authd.mcs
PCCUserAuth	User authentication	auth.mcs
LnaMau	LNA data types definition	lnadata.mcs
LnaMauIf	LNA MAU interface	lnaif.mcs

C.2.7 Data types General

```
DATA TYPES General
  * This module defines a set of general data-types. They are all defined as
global.

  * Revision history:
    990831  1.1  Second official release

  VERSION     1.1
  DATE        1999-08-31
  RESPONSIBLE IEC TC80/WG6

REFERENCES
  none

;-------------------------------------------------------------
USAGE
  * This file contains general definitions for PFS

GLOBAL

;-------------------------------------------------------------
INTERPRETATION FieldOk OF bool_m
  * Each bit indicates whether the corresponding field in a block is
    valid. Valid means interpretable, i.e., a null-terminated string
    can be flagged as valid A zero length variable length array can
    alos be valid. Fields are counted from the top of the data block
    definition (as index zero) and down (increasing index). The field
    is valid if the corresponding bit is TRUE (1).

;-------------------------------------------------------------
INTERPRETATION BlockOk OF bool_m
  * Value indicates whether data block is valid or not.

  FALSE               ; Block is illegal
  TRUE                ; Block is ok

;-------------------------------------------------------------
DATA BLOCK Time
  * A representation of relative time. Note that the representation
    cannot accomodate longer time differences than approximately 136
    years.

  word32_m    sec
  word32_m    usec  ; Micro-seconds fraction of above

;-------------------------------------------------------------
INTERPRETATION GlobalTime OF Time
  *  GlobalTime contains number of atom clock seconds and micro-seconds
     since midnight 1. January 1970 UTC, *excluding* leap
     seconds. This is compatible with POSIX time representation (for
     use in, e.g., ctime or localtime functions). This time
     measurement has an offset to UTC time that is changed for each
     (negative or positive) leap second.

  * Note 2: The constant UTCOFFSET_JAN1999 can be used as an
    approximation to the offset to UTC time. One should keep in mind
    that there may be a second or so difference between the MiTS time
    as specified by the system server and real UTC time. This should
    be no problem as long as this is consistent during a voyage.

  * Note 3: The GPS satelite or a radio or modem based time standard
    server can be used to update the offset count.

  * Note 4: When using POIX time conversion functions, one should
    check if the leap second count is taken into consideration (it
    should not be) and if the second count is signed or not. A signed
    second count will cause problems around year 2038 when a signed
    seconds count wrap around to negative.

  * Note 5:  The second counter wraps around sometime in the year
    2107. Use of the time structure should take care to do modulo
    arithmetic correctly.
```

```
  sec   ; Seconds since 1970-01-01 00:00:00 UTC
  usec  ; Micro-seconds fraction of above

-------------------------------------------------------------
CONSTANT UTCOFFSET_JAN1999 IS 22
  * The last leap second (at date of writing) occured at the start of
    January 1999. The total offset between POSIX time and UTC was after
    that event 22 seconds. A complete list of leap seconds can be
    found below. The list is adapted from a file located at
    ftp://maia.usno.navy.mil/ser7/leapsec.dat.

    * 1972 JAN  1
    * 1972 JUL  1
    * 1973 JAN  1
    * 1974 JAN  1
    * 1975 JAN  1
    * 1976 JAN  1
    * 1977 JAN  1
    * 1978 JAN  1
    * 1979 JAN  1
    * 1980 JAN  1
    * 1981 JUL  1
    * 1982 JUL  1
    * 1983 JUL  1
    * 1985 JUL  1
    * 1988 JAN  1
    * 1990 JAN  1
    * 1991 JAN  1
    * 1992 JUL  1
    * 1993 JUL  1
    * 1994 JUL  1
    * 1996 JAN  1
    * 1997 JUL  1
    * 1999 JAN  1

;-------------------------------------------------------------
DATA BLOCK Version
  * Version codes generally used in this protocol. An increment in major
    number indicates a specification (protocol) change that may render
    older versions incompatible with newer. Downward compatibility may
    be supported but shall not be relied on.  The value zero for major
    represent a test version with unknown relationship to official
    versions. An increment in minor number represents some form of
    correction or minor adjustment that retains upwards specification
    compatibility. An increment in release number indicates software
    fixes with possible changes in functionality only to correct
    previous errors. The date represents the compilation date and time
    for the software implementing the version.

  word16_m    major
  word16_m    minor
  word16_m    release
  GlobalTime  date

;-------------------------------------------------------------
INTERPRETATION EngineeringUnit OF word16_m
  * Specification of scalar dimension. Most units are based on SI, but
    some special considerations have been made to naval units (knots,
    nautical miles and angualar measures).

  * Note: Different storage classes can use the same engineering unit,
    e.g., a time can be represented in a float or an integer. It is
    usually necessary to know both storage class and engineering unit
    to interpret a number.

  * EU_NAVDIRECTION is used both for positions (E/W or N/S) or compass
    headings. The context defines how it is used.

  0 = EU_OTHER          ; other (use description)
  1 = EU_UNKNOWN        ; not known
                        ;
  2 = EU_COUNT          ; number - dimension-less
  3 = EU_RATIO          ; ratio - dimension-less
                        ;
```

```
  4 = EU_TEXT           ; No unit - text string
                        ;
  10 = EU_LENGTH        ; m (linear measure)
  12 = EU_AREA          ; m**2 (area)
  13 = EU_VOLUME        ; m**3 (volume)
  14 = EU_NAVDISTANCE   ; nautical miles - 1852 m (navigational length )
                        ;
  20 = EU_ANGLE         ; radians (angular measure)
  21 = EU_NAVDIRECTION  ; degrees, positive clockwise (angualar
                        ; measure, normally N is zero).
  22 = EU_POSITION      ; tuple: first is degrees N positive (S neg),
                        ; degrees E positive (W neg) (position in
                        ; latitude, longitude).
                        ;
  30 = EU_VELOCITY      ; m/s (linear velocity)
  31 = EU_NAVVELOCITY   ; knots - nm/h (navigational velocity)
  32 = EU_ANGVELOCITY   ; radians/s (angular velocity)
                        ;
  40 = EU_ACCEL         ; m/s**2 (linear acceleration)
  41 = EU_ANGACCEL      ; radians/s**2 (angular acceleration)
                        ;
  50 = EU_TIME          ; s (time)
  51 = EU_FREQUENCY     ; Hz (frequency)
                        ;
  60 = EU_MASS          ; kg (mass)
  61 = EU_DENSITY       ; kg/m**3 (density)
  62 = EU_MASSFLOW      ; kg/s (mass flow)
                        ;
  70 = EU_FORCE         ; N (Force)
  71 = EU_TORQUE        ; Nm (Torque)
  72 = EU_PRESSURE      ; Pa or N/m (Pressure)
                        ;
  80 = EU_ENERGY        ; J (Energy)
  81 = EU_POWER         ; W (Power)
  82 = EU_HEATFLOW      ; J/m**2 (Heatflow)
  83 = EU_TEMP          ; Degrees C (temperature)
  84 = EU_ABSTEMP       ; Degrees K (absolute temperature)
                        ;
  90 = EU_DYNVISC       ; N/m * s (Dynamic viscosity)
  91 = EU_KINVISC       ; m**2/s (Kinematic viscosity)
                        ;
  100 = EU_RESISTANCE   ; Ohm (Resistance)
  101 = EU_CURRENT      ; A (Current)
  102 = EU_VOLTAGE      ; V (Voltage)
                        ;
  1000 = EU_RAW         ; Raw byte stream (file)
  1001 = EU_JAVA        ; Java text code
  1002 = EU_TCLTK       ; TCL/TK code
  1003 = EU_EXPRESS     ; EXPRESS text code
                        ;
  10000 = EU_FREE       ; User defined from this value and up

;-------------------------------------------------------------
INTERPRETATION LocationCode OF word32_m
  * Location codes. Unused codes are user defined. For future expansion
    these codes should start at USERLOCATIONS.

  0 = LC_EVERYWHERE
  ;
  100000 = LC_USERLOCATIONS   ; Above and including this number

;-------------------------------------------------------------
INTERPRETATION TPnet OF word16_m
  * Code for various T-profile network (IEC 61162-420).

   0 = TPN_ANYADR_USER  ; User specified
   1 = TPN_MAUADR_IPC   ; MAU-LNA system specific IPC
   2 = TPN_MAUADR_TCP   ; MAU-LNA over WAN TCP
 129 = TPN_LNADR_IPV4R  ; IPV4 Std. LNA-LNA redundant
 130 = TPN_LNADR_IPV4RE ; IPV4 Extended LNA-LNA redundant
 131 = TPN_MMADR_IPV4R  ; IPV4 Std. MAU-MAU redundant
 132 = TPN_MMADR_IPV4RE ; IPV4 Extended MAU-MAU redundant
 140 = TPN_LNADR_IPV4   ; IPV4 Std. LNA-LNA non-redundant
 141 = TPN_LNADR_IPV4E  ; IPV4 Extended LNA-LNA non-redundant
 142 = TPN_MMADR_IPV4   ; IPV4 Std. MAU-MAU non-redundant
```

```
 143 = TPN_MMADR_IPV4E  ; IPV4 Extended MAU-MAU non-redundant

;-------------------------------------------------------------
DATA BLOCK address_m
  * Address of a stream interface. This consists of a T-profile type and a
    T-profile dependent address block. The currently defined address blocks are:
      TPN_ANYADR_USER:  User defined, any length
      TPN_MAUADR_IPC:   System dependent code, e.g., process id
      TPN_MAUADR_TCP:   Void, zero length
      TPN_LNADR_IPV4R:  Void, zero length
      TPN_LNADR_IPV4RE: 8 octets, two word32_m Internet addresses
      TPN_MMADR_IPV4R:  Void, zero length
      TPN_MMADR_IPV4RE: 8 octets, 2 word32_m
      TPN_LNADR_IPV4:   Void, zero length
      TPN_LNADR_IPV4E:  4 octets: word32_m
      TPN_MMADR_IPV4:   Void, zero length
      TPN_MMADR_IPV4E:  4 octets: word32_m

  TPnet                  tProfile   ; T-profile in use
  [word8_m:48]word8_m    tAddress   ; The address

;-------------------------------------------------------------
INTERPRETATION TPService IS word32_m
  * A TP network service class. Values defined by T-profile.

  0 = TPNS_UNKNOWN

;-------------------------------------------------------------
INTERPRETATION TPSInstance IS word32_m
  * A TP network service class instance. Values depedent on service. For
    MAU-MAU stream over Internet, the value is TCP port number.

  0 = TPNSC_NONE
```

C.2.8 Application PACSimpleApplication

```
APPLICATION PACSimpleApplication

  * This application contains the general framework for the creation
    of a minimum PISCES application.

  * Revision history :
    1999-08-31 1.1 ojr, SINTEF: For IEC
    1998-12-01 1, ojr : First

  VERSION 1.1
  DATE 1999-08-31
  RESPONSIBLE IEC TC80/WG6

;-------------------------------------------------------------
REFERENCES

  INTERFACE PCCVersionCodes

;-------------------------------------------------------------
USAGE

  * Minimum required functionality for PFS application.

;-------------------------------------------------------------
INTERFACES

  ACCEPT Application

    INTERFACE COMPONENT PCCVersionCodes
```

C.2.9 Application PACFullApplication

```
APPLICATION PACFullApplication DERIVED FROM PACSimpleApplication

  * This application contains the general framework for the creation
    of a complete PISCES application.

  * Revision history :
    1999-08-31 1.1: For IEC, ojr, SINTEF
    1998-12-01 1, ojr : First

  VERSION 1.1
  DATE 1999-08-31
  RESPONSIBLE IEC TC80/WG6

;-------------------------------------------------------------
REFERENCES

  INTERFACE PCCApplicationInfo

;-------------------------------------------------------------
USAGE

  * Minimum functionality for full application.

;-------------------------------------------------------------
INTERFACES

  ACCEPT Application

    INTERFACE COMPONENT PCCApplicationInfo
```

C.2.10 Interface PCCVersionCodes

```
INTERFACE PCCVersionCodes
  * This interface gives access to basic information about the
    application.

  * Revision history :
    1999-08-31 1.1, ojr: For IEC
    1998-12-01 1, ojr: Change name
    1998-08-26 C, ojr: Only version codes - rest in application
    1998-08-13 B, ojr : Modified version codes, operating with
    interface components, connection point numbers. Use general data
    types as reference.
    1998-07-23 A, smt : created

  VERSION     1.1
  DATE        1999-08-31
  RESPONSIBLE IEC TC80/WG6

;-------------------------------------------------------------
REFERENCES
  General

;-------------------------------------------------------------
USAGE
  This interface is for general use by all applications
  compliant to the PISCES companion standard. It is also
  contained in the application base class defined within the
  PISCES foundation classes

  *NOTE: All variable length strings are length encoded and not null
  terminated.

;-------------------------------------------------------------
DATA TYPES

  ;-----------------------------------------------------------
  INTERPRETATION PFClass OF word16_m
    * PFC application class code.

    0 = PFC_NONE             ; Special class
    1 = PFC_LNA              ; LNA MAU
    2 = PFC_FULL             ; Full server
    3 = PFC_SIMPLE           ; Simple interface

;-------------------------------------------------------------
CONNECTION POINTS

  ;-----------------------------------------------------------
  FUNCTION GetApplicationInfo
   * This connection point returns general information about
     the application concerned. This includes header information (the
     name of the responsible author, manufacturer and type of
     application) as well as information about the interfaces.

   * Post condition: Returns number of interfaces supported in
     addition to this one. I.e., zero is in principle a legal
     number. The number specifies interface components. Interface
     components are numbered from zero (this one) to
     interfaces.

    INPUT
      none

    OUTPUT
      [word16_m:64]char8_m manufacturer    ;manufacturer
      [word16_m:64]char8_m model           ;model (type)
      int32_m     interfaces               ;number of additional interfaces
      PFClass     class           ;PFC type
      Version     appVersion      ;Application revision
      Version     protoVersion    ;Protocol revision
```

C.2.11 Interface PCCApplicationInfo

```
INTERFACE PCCApplicationInfo
  * This interface gives access to information about
    applications (general attributes), application interfaces
    and their connection points

  * Revision history :
    1999-08-31, 1.1, ojr, For IEC
    1998-12-01 1, ojr : Changed name
    1998-08-26 C, ojr : Removed basic version codes to version.mcs
    1998-08-13 B, ojr : Modified version codes, operating with
    interface components, connection point numbers. Use general daat
    types as reference.
    1998-07-23 A, smt : created

  VERSION     1.1
  DATE        1999-08-31
  RESPONSIBLE IEC TC80/WG6


;-------------------------------------------------------------
REFERENCES
  General

;-------------------------------------------------------------
USAGE
  This interface is for general use by all applications
  compliant to the PISCES companion standard. It is also
  contained in the application base class defined within the
  PISCES foundation classes

  *NOTE: All variable length strings are length encoded and not null
  terminated.

;-------------------------------------------------------------
CONNECTION POINTS

  ;-----------------------------------------------------------
  FUNCTION GetInterfaceInfo
   * This command is used to retrieve detailed information
     about one selected interface component in the application
     concerned.

   * Precondition :
     Input parameter interface (see below) must be a number of an
     interface, numbered from zero (this interface itself) to
     interfaces (from GetApplicationInfo).

   * Postcondition :
     Information about the interface is retrieved. The data is valid
     if the flag isOk is set to TRUE. Please note that the following
     fields are only valid for an accept interface (i. e. type == 1):
           - auth_flag
           - numb_clients
           - password
     The field priority is only valid for a connect interface
     (i.e. type == 0).

     * Number of connection points are the total number of interfaces,

     * Interface number is negative for illegal interfaces:
       -1: No such interface
       -2: No data on interface

    INPUT
      int32_m     interface

    OUTPUT
      int32_m     interface   ; Interface number, as input
      [word16_m:32]char8_m appName   ; Name of interface
      [word16_m:32]char8_m className ; Name of interface class
      bool_m   isAccept       ; FALSE: connect / TRUE: accept
      bool_m   hasAuth        ; authentication necessary
```

```
    bool_m   needPassword   ;interface password protected (TRUE/FALSE)
    word16_m acceptClients  ;max. number of clients
    word16_m max_mess_rate  ;maximum message rate possible
    word16_m min_mess_rate  ;minimum message rate allowed
    word16_m priority       ;LOW, MEDIUM or HIGH
    int32_m  noCPs          ;Number of connection points
    Version  vNo            ;Version information

;-------------------------------------------------------------
FUNCTION GetCPInfo
 * This command is used to retrieve detailed information
   about one selected connection point.

 * Precondition :
   Input parameter interface (see below) must be a valid code for an
   existing interface component. conn_pt be a valid code for a
   connection point belonging to the interface.

 * Postcondition :
   Information about the connection point conn_pt is retrieved. The
   data is valid if returned interface and conn_pt is
   non-negative. Negative values for one or both are: -1: No such
   entry for this interface/conn_pt -2: No data available on this
   interface/conn_pt

  INPUT
    int32_m interface
    int32_m conn_pt

  OUTPUT
    int32_m interface
    int32_m conn_pt
    [word16_m:1950]char8_m   format  ;Format string
    [word16_m:32]char8_m     name    ;Name of CP
```

C.2.12 Data types UserAuth

```
DATA TYPES UserAuth
  * These data types are related to user authentication.

  * Revision history :
    1999-08-31, 1.1, ojr, For IEC
    1998-12-01 1, ojr : First

  VERSION     1.1
  DATE        1999-08-31
  RESPONSIBLE IEC TC80/WG6


;-------------------------------------------------------------
REFERENCES

  DATA TYPES General


;-------------------------------------------------------------
GLOBAL

  ;-------------------------------------------------------------
  INTERPRETATION UserGroup OF int32_m
    * Codes for user groups.

    0 = UG_CAPTAIN
    1 = UG_CHIEF
    2 = UG_DECKOFFICER
    3 = UG_OOW            ; Officer On Watch
    4 = UG_OTHERENGINEER  ; not chief
    ;
    1000 = UG_USER        ; user defined from here

  ;-------------------------------------------------------------
  INTERPRETATION ConsoleGroup OF int32_m
    * Codes for operating consoles (from IMO A.830(19) and other)

    0 = CG_OTHER      ; Not defined location
    1 = CG_BRIDGE     ; Navigation bridge
    2 = CG_MACHINERY  ; Machinery control room
    3 = CG_FIRE       ; Central fire control station
    4 = CG_LOCAL      ; At location of equipment being monitored
    5 = CG_ENGINEER   ; Engineer's accomodation
    ;
    6 = CG_BRIDGEWINGP ; Navigation position port
    7 = CG_BRIDGEWINGS ; Navigation position starboard
    8 = CG_BOWCONTROL  ; Navigation and DP position
    9 = CG_BOWDOOR     ; For bow door
    ;
    1000 = CG_USER   ; User defined position from this code


  ;-------------------------------------------------------------
  INTERPRETATION UaStatus of int32_m
    * Return status for request to authenticate.

    0 = US_ALLOWED
    1 = US_WRONGUSER
    2 = US_WRONGCONSOLE
    3 = US_WRONGSPECIFICCONSOLE
    4 = US_WRONGSPECIFICUSER
    5 = US_WRONGPASSWORD
    6 = US_UNKNOWN          ; Other unknown error
    ;
    1000 = US_OTHER         ; Other specific errors

  ;-------------------------------------------------------------
  INTERPRETATION UaCode of [32]bool_m
    * The user authentication code. Increasing codes give increasing
      levels of authorities. Coding is dependent on controlled
      application May be a bit-map?
```

```
    0 = UA_NONE   ; No secure operation is allowed when all bits zero
    1 = UA_READ   ; Allowed to read data and attributes
    2 = UA_WRITE  ; Allowed to write some data
    3 = UA_WATTR  ; Allowed to write and set some attributes
    4 = UA_ALARM  ; Allowed to acknowledge some alarms
    ;
    0xffffffff = UA_ALL ; All operations allowed when all bits set

  ;-------------------------------------------------------
  INTERPRETATION CommandCode OF int32_m
    * Codes used in the function calls.

    0 = CC_NOOP         ; No operation
    1 = CC_REQCONSOLE   ; Request transfer of console to remote
    2 = CC_SETCONSOLE   ; Force to be set as console
    3 = CC_ACKTRANSFER  ; Acknowledge transfer
    4 = CC_WHATCONSOLE  ; What is console
    ;
    1000 = CC_USER      ; Other commands
```

C.2.13 Interface PCCUserAuth

```
INTERFACE PCCUserAuth
  * This interface is used for user authentication. A client asks a
    server for authentication codes based on a passwords, a user code
    the code for the controlling function and the function to control.

  * Revision history :
    1999-08-31 1.1, ojr, For IEC
    1998-12-01 1, ojr : Removed data types, changed name
    1998-08-26 C, ojr : Added console change and timeout on auth
    1998-07-23 B, ojr : modified
    1998-07-23 A, smt : created

  VERSION     1.1
  DATE        1999-08-31
  RESPONSIBLE IEC TC80/WG6


;-------------------------------------------------------------
REQUIRED DOCUMENTATION

  PASSWORDS  ; It must be documented how passwords are configured in
             ; both client and server

;-------------------------------------------------------------
REFERENCES

  DATA TYPES General
  DATA TYPES UserAuth

;-------------------------------------------------------------
CONNECTION POINTS

  ;-----------------------------------------------------------
  FUNCTION Authenticate
   * This connection point is used to identify a user and returns an
     authentication code. The protocol will ensure that it is
     possible to identify the client MAU between connection points and
     interfaces.

     * Precondition: Input user and console codes and password. Input
       optionally a specific user and console code. This may be
       required by some applications. A new request terminates
       automatically any previous authorisations.

     * Postcondition: Authorisation status. The requesting MAu will
       automatically have an authorisation level corresponding to user
       and console. This will be used where applicable on other
       requests to the server MAU. The time field gives optionally a
       time to live for the authorisation (zero is infinite). A new
       request must be sent before this time to validate the
       authorisation. The authorisation code and descriptive string
       specifies allowed operations.

    INPUT
      UserGroup       user         ; What user code
      ConsoleGroup    console      ; What control position
      [8]char8_m      spConsole    ; Optional specific console code or null
      [8]char8_m      spUser       ; Optional specific user code or null
      [8]char8_m      password     ; Password

    OUTPUT
      UaStatus        status       ; return code
      UaCode          authCode     ; authorisation code
      Time            ttl          ; Time that authorisation is valis
      int32_m         console      ; Console number
      [32]char8_m     authDesc     ; Optional descriptive string

  ;-----------------------------------------------------------
  SUBSCRIBE IsControl
   * This subscription point informs all connected consoles about
     current status.
```

```
    * Precondition: The listeners console code must be established by
      doing an authorisation request. If not, the lsitner should
      assume zero as this code never is used.

    * Postcondition: The command code specifies necessary actions:
      REQCONSOLE and nextConsole is self: Acknowledge take over
      SETCONSOLE: Register new console in command, use provided data.
      WHATCONSOLE: Information about new or changed console
      configuration. Use provided data.
      Other: ignore
    * The user, console, spConsole and spUser describes the
      attributes of the new console when SET, REQ or WHAT
      CONSOLE. Invalid else.

    OUTPUT
      CommandCode     command      ; Command to consoles
      int32_m         inConsole    ; Current console in command
      int32_m         nextConsole  ; Next to be console in command
      UserGroup       user         ; What user code
      ConsoleGroup    console      ; What control position
      [8]char8_m      spConsole    ; Optional specific console code or null
      [8]char8_m      spUser       ; Optional specific user code or null

;-------------------------------------------------------------
FUNCTION SetControl
 * This connection point does one of several things:
   - Inquire about current console in command
   - Ask for transfer to another console
   - Ask for forced transfer to this console
   - Acknowledge transfer to this console
   by the character string new_console. It returns the name of
   the workstation previously controlling the process as well as a
   flag indicating whether the assignment has been executed
   successfully.

 * Precondition: Authorisation must have been established. A console
   code for own console will be returned through that. One can ask
   for transfer to own or to another. One can also ask for console
   in command (do not usually require authorisation).

  INPUT
    ControlCommand  command      ; Operation to be performed
    int32_m         nextConsole  ; Myself or transfer to

  OUTPUT
    ControlStatus   status
    int32_m         nextConsole  ; Requested console number
```

C.2.14 Data types LnaMau

```
DATA TYPES LnaMau
  * This module defines a set of general data-types to be used in PISCES
    companion standard specifications for the LNA-MAU.

  * Revision history:
    1999-08-31 V1.1 Some changes in version and address (ojr, SINTEF)
    1998-11-30 V0.1 Transcription to PISCES CS format (jhe - ISSUS)


  VERSION     1.1
  DATE        1999-08-31
  RESPONSIBLE IEC TC80/WG6


;-----------------------------------------------
REFERENCES
  none

;-----------------------------------------------
GLOBAL

;CONSTANTS:

CONSTANT CONST_ELEM_CPLIST IS 12
   ;The number of elements in a connection point list.

CONSTANT CONST_ELEM_LNALIST IS 40
   ;The number of elements in a lna list.

CONSTANT CONST_ELEM_MAULIST IS 45
   ;The number of elements in a mau list.

CONSTANT CONST_FORMAT_LENGTH IS 1500
   ;The max. string length of a format string.

CONSTANT CONST_CPNAME_LENGTH IS 32
   ;The max. string length of a connection point name.

CONSTANT CONST_IFNAME_LENGTH IS 32
   ;The max. string length of an interface name.

CONSTANT CONST_LNANAME_LENGTH IS 32
   ;The max. string length of a lna name.

CONSTANT CONST_MAUNAME_LENGTH IS 32
   ;The max. string length of a mau name.

;-----------------------------------------------
;INTERPRETATIONS:

INTERPRETATION CPStatus OF word8_m
  ;The definition of type CPStatus. Variables of this type may have
  ;the following values:

    0 = undefined
    1 = C_FIND_MAU
    2 = C_FIND_MC
    3 = C_OPEN

INTERPRETATION CPType OF word8_m
  * This variable contains the service type of the connection point
    (eg. SUBSCRIBE, READ, WRITE)

    0 = undefined
    1 = READ
    2 = WRITE
    3 = FUNCTION
    4 = SUBSCRIBE
    5 = NONACKED_WRITE
    6 = BROADCAST
    7 = INDIVIDUAL SUBSCRIBE
```

```
;-----------------------------------------------
;DATA BLOCKS:

DATA BLOCK LnaStatus
   ;The data block LnaStatus has the following elements:

   address_m address ; The network address of LNA.
   word32_m  networkNode    ; Node number of ditto
   int8_m    status
       * The status of the LNA or CNA.
         Possible values:
    0 = undefined/ not running
    1 = LNA_GOT_STATUS
    2 = LNA_WAIT_FOR_STATUS
    3 = LNA_REGISTERING
    4 = LNA_REGISTERED

   Version       versionNo
       ; Software version number and compilation date.

DATA BLOCK MauInfo
   ;The data block MauInfo has the following elements:

   [CONST_MAUNAME_LENGTH]char8_m   mauname
       ; The MAU name.

   int8_m  status
       * The status of the MAU.
         Possible values:
    0 = undefined status
    1 = ACTIVE

   word32_m        numberConnectCPs
       ; Number of established CONNECT connection points of this MAU.

   word32_m        numberAcceptCPs
       ; Number of established ACCEPT connection points of this MAU.


DATA BLOCK ACPInfo
   ;The data block ACPInfo has the following elements:

   [CONST_CPNAME_LENGTH]char8_m    cpname
       ; The name of the connection point.

   [CONST_IFNAME_LENGTH]char8_m    ifname
       ; The name of the interface of this connection point.

   CPType  cpType
       * This variable contains the service type of the connection point
    (eg. SUBSCRIBE, READ, WRITE).


DATA BLOCK CCPInfo
   ;The data block CCPInfo has the following elements:

   [CONST_CPNAME_LENGTH]char8_m    cpname
       ; The name of the connection point.

   [CONST_IFNAME_LENGTH]char8_m    ifname
       ; The name of the interface of this connection point.

   [CONST_MAUNAME_LENGTH]char8_m   rmau
       ; The name of the remote (server) MAU.

   CPStatus status
       ; The status of the connection point.

   CPType  cpType
       * This variable contains the service type of the connection point
         (eg. SUBSCRIBE, READ, WRITE).
```

C.2.15 Interface PCCLNAStats

```
INTERFACE PCCLNAStats
  * This is the LNA-MAU interface.

  * Revision history
    1998-12-01: First, jhe, ISSUS

  VERSION     1.1
  DATE        1999-08-31
  RESPONSIBLE IEC TC80/WG6


;-----------------------------------------------

REFERENCES
  DATA TYPES LnaMau

USAGE
  This Companion Standard Description provides the interface to the
  LNA-MAU. It is supposed to be a part of each LNA.


REQUIRED DOCUMENTATION
  LnaMau

;-----------------------------------------------

CONNECTION POINTS

READ GetStatus
  * This connection point shall be used to retrieve status information
    about the LNA from the LNA-MAU.

OUTPUT
  LnaStatus    lnaStatus
    ; The status of the LNA.

  word32_m numberOfLocalMAUs
    ; Number of local MAUs.

  word32_m numberOfRemoteLNAs
    ; Number of remote LNAs.

;-----------------------------------------------
FUNCTION GetMcpList
  * This connection point shall be used to retrieve a list of MCPs of a
    given MAU. More than one list may exist. Each list can be addressed
    by the listNumber attribute.

INPUT
  word32_m  listNumber
    ; The number of the requested list (zero-based).

  [CONST_MAUNAME_LENGTH]char8_m   mauname
    ; The MAU name.

OUTPUT
  [word32_m:CONST_ELEM_CPLIST]ACPInfo     acceptList
    ; The list of accept CPs.

  [word32_m:CONST_ELEM_CPLIST]CCPInfo     connectList
    ; The list of connect CPs.

  word32_m totalNumberOfLists
    ; The total number of lists currently to be retrieved.

  word32_m listNumber
    ; The number of this list (zero-based).

  word32_m listReference
    * The current reference number. The reference is changed each time
      the status or number of connection points have changed.
      Use this variable to identify changes while retrieving
```

```
      a MCP list.


;-----------------------------------------------
FUNCTION GetMauList
  * This connection point shall be used to retrieve a list of MAUs attached
    to this LNA. More than one list may exist. Each list can be addressed
    by the listNumber attribute.


INPUT
  word32_m listNumber
    ; The number of the requested list (zero-based).

OUTPUT
  [word32_m:CONST_ELEM_MAULIST]MauInfo    list
    ; The list of MAUs.

  word32_m totalNumberOfLists
    ; The total number of lists currently to be retrieved.

  word32_m listNumber
    ; The number of this list (zero-based).

  word32_m listReference
    * The current reference number. The reference is changed each time
      the status or number of connection points have changed.
      Use this variable to identify changes while retrieving
      a MAU list.

;-----------------------------------------------
FUNCTION GetLnaList
  * This connection point shall be used to retrieve a list of remote LNAs known
    to this LNA. More than one list may exist. Each list can be addressed
    by the listNumber attribute.


INPUT
  word32_m listNumber
    ; The number of the requested list (zero-based).

OUTPUT
  [word32_m:CONST_ELEM_LNALIST]LnaInfo    list
    ;The list of LNAs.

  word32_m totalNumberOfLists
    ; The total number of lists currently to be retrieved.

  word32_m listNumber
    ; The number of this list (zero-based).

  word32_m listReference
    * The current reference number. The reference is changed each time
      the status or number of connection points have changed.
      Use this variable to identify changes while retrieving
      a LNA list.

;-----------------------------------------------
SUBSCRIBE SubscribeToLnaChanges
  * This subscribe connection point can be used to get notified in case of a
    change of the status or one of the lists of the LNA.

OUTPUT
  int8_m  lnaCnaStatusChanged
    * The variable may have the following values:
      0 = unchanged
      1 = The status of the LNA or CNA has changed (use GetStatus)

  int8_m  mauListChanged
    * The variable may have the following values:
      0 = unchanged
      1 = The MAU list has changed (use GetMauList)

  word32_m numberOfChangedMauList
    * The number of the first MAU list that has changed. Only valid if
```

```
      mauListChanged is not "unchanged".

  int8_m  lnaListChanged
    * The variable may have the following values:
      0 = unchanged
      1 = The LNA list has changed (use GetLnaList)

  word32_m numberOfChangedLnaList
    * The number of the first LNA list that has changed. Only valid if
      lnaListChanged is not "unchanged".

  int8_m  mcpListChanged
    * The variable may have the following values:
      0 = unchanged
      1 = The MCP list has changed (use GetMcpList)

  word32_m numberOfChangedMcpList
    * The number of the first MCP list that has changed. Only valid if
      mcpListChanged is not "unchanged".

  [CONST_MAUNAME_LENGTH]char8_m   mauChanged
    ; The name of the MAU which MCP has changed.

;-----------------------------------------------
FUNCTION GetFormatString
  * This connection point may be used to retrieve the format string
    of a particular MCP of a MAU from the LNA-MAU.

INPUT
  [CONST_MAUNAME_LENGTH]char8_m   mauname
    ; The name of the MAU to be addressed.

  [CONST_CPNAME_LENGTH]char8_m    cpname
    ; The name of the connection point to be addressed.

  bool_m  acceptFlag
    * This flag indicates weather the MCP is of
      an ACCEPT or CONNECT type.
      FALSE = CONNECT
      TRUE = CONNECT

  CPType  cpType
    * This variable contains the service type of the connection point
      (eg. SUBSCRIBE, READ, WRITE).

OUTPUT

  [CONST_FORMAT_LENGTH]char8_m    format
    ; The format string of this connection point.
```

Annex D
(Normative)

General alarm and monitoring companion standards

D.1 Introduction and general principles

This document contains the companion standards for a general interface to automation and alarm systems. The standards are also useful in the context of navigation, where they can be used for interfaces to alarm system or to higher level decision support systems.

These companion standards are based on the manipulation of information based on tag names. The tag name is an identifier for the information item. The term *tag* will in the following be used synonymously with *information item*. Each tag is associated with a value, possibly an alarm state and a set of attributes.

The companion standards provide the following general mechanisms:

a) Search for tags on a specific MAU.

b) Search for tags on the network (via anonymous broadcast).

c) Reading and writing tag values.

d) Reading and writing tag attribute values.

e) Subscribing on values from individual or set of tags.

f) Mechanisms for subscribing to and acknowledging alarms.

Access to tags can optionally be associated with user or workstation authentication. This is normally necessary for alarm acknowledgement and value writing.

The companion standards described here are application independent. The set of tag names will determine what application is associated with a certain MAU.

D.2 Definitions

Tag name: A text string identifying an information item. Several tag names can reference the same information item.

Tag number: A code referencing one information item. There is a one to one relationship between tag number and the information item.

Yard tag: A tag name (usually) assigned by the yard. The yard tag is normally associated with a physical device, e.g., a temperature transmitter, and can in some cases also be associated to a representative information item (tag number), e.g., the temperature measurement.

D.3 Functionality overview

D.3.1 Companion standard for tag based monitoring and alarm system

The general idea is that all PISCES applications of a certain class should supply one uniform service to other PISCES clients that allows the clients to read and write data from or to the server in a standard way. This mechanism consists of a companion standard that defines certain general data objects for reading and writing based on "tag names".

By having such a system one can generate application independent code that can be used by any client to read and write to any server. This is particularly appropriate for "decision support"

type or "data fusion" type applications that lie on a higher abstraction level than the basic control applications. It is still assumed that the control applications have more specialised function blocks for use in between them, e.g., for GPS data output.

D.3.2 Client-server architecture

The principle for use of these companion standards is that a server makes information items available to any client in the system. Likewise, given that the appropriate functionality is installed, the client can write to information entities or subscribe to updates. It is assumed that the server does not need to know the clients a priori and that the system is based on a client-server architecture.

D.3.3 Tag number

Each information item in one server is identified by a tag number. This number is unique for that server. To each tag number there can be several tag names. This can be used to provide the same information item with different names based on yard naming principles (yard tags), standard naming principles and/or manufacturer dependent naming principles.

The basic interface component provides functionality for mapping tag names to tag numbers. This is through a search function with functionality dependent on the server in question. Very simple servers may just have a fixed set of tag numbers with a static mapping to a set of tag names. These servers may not support any search functional at all.

Tag numbers are the identifiers used when tag related information is transferred to and from the server.

D.3.4 Tag sets

Servers that support subscribe and alarm handling do this with the help of tag sets. Tag set are also established through searches, but can be manipulated with the help of functions in the `PCCTagSet` interface.

Subscriptions on values or alarms can only be done through tag sets.

D.3.5 Tag information

Each tag is associated with a set of information attributes. These can be retrieved through the MCP `GetTagInfo`. This information is, e.g., the engineering unit, expected precision, sampling interval, a description string etc. It is expected that this information is static through out the tags life.

D.3.6 Tag attributes

Some tags can have attributes associated with them. These are semi-static information that can change through the tags life, but not very often and rarely or never uncontrolled, e.g., filter constants, alarm limits, scaling factors etc. The number of attributes can be read in the tag information data structure and the attributes can be inspected and changed through the attribute related interface components.

D.3.7 Tag data

Each tag has a data value associated with it that is expected to change more or less continuously. The data value can be read, written or subscribed to through the relevant interface components.

Read data will always have a quality flag saying to what degree the value has the required quality, a time stamp and pending alarm flag information.

D.3.8 Alarms

Alarms are a special case of tags where the server provides a mechanism for keeping a watch on the tag value to raise an exception if the value changes in some defined manner. Alarms are also assigned a sequence number that allows the client to determine if more than one alarm has been raised on one tag.

Alarms needs to be acknowledged before they are removed from the active alarm list. This is the case even if the value goes back to normal.

D.4 Application classes

The companion standards define three different application classes, all derived from `PACFullApplication`. These classes are shown below.

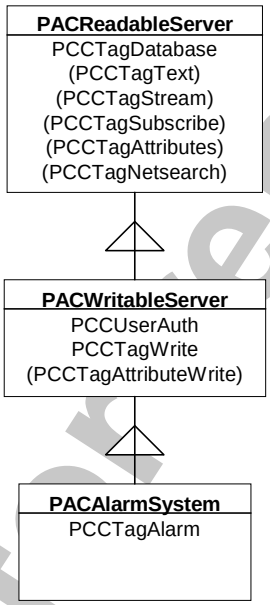

Figure 16 - Tag application classes

Each application class have certain capabilities that can be added to and thereby specialising the class into a more advanced class:

a) `PACReadableServer`: This is the basic class with capabilities for searching for tags and for reading values, text or floating point. It can be extended with component interfaces for subscription and for reading attribute values. It can also be extended with a component interface for network wide search for tags.

b) `PACWritableServer`: This is an extension that allows writing tags and optionally attributes. This application class requires user authentication.

c) `PACAlarmSystem`: This extends the write-able server with a subscription on alarms and a mechanism for acknowledging alarms.

Note that both the subscription and the alarm handling component interfaces require the use of the set definition interface component.

All interface components except the user authentication component shall be defined as belonging to the physical interface “`Tags`”.

D.5 Companion standard structure

The companion standards are organised in several component interfaces as discussed in the previous clause. In addition to the interface components, the definitions also contains the application definitions and a general data type definition.

The below diagram shows the relationship between the interface components. Note that the three classes `TagRead`, `TagWrite` and `TagAlarm` represent the application classes as discussed in the previous clause and not interface classes as such.

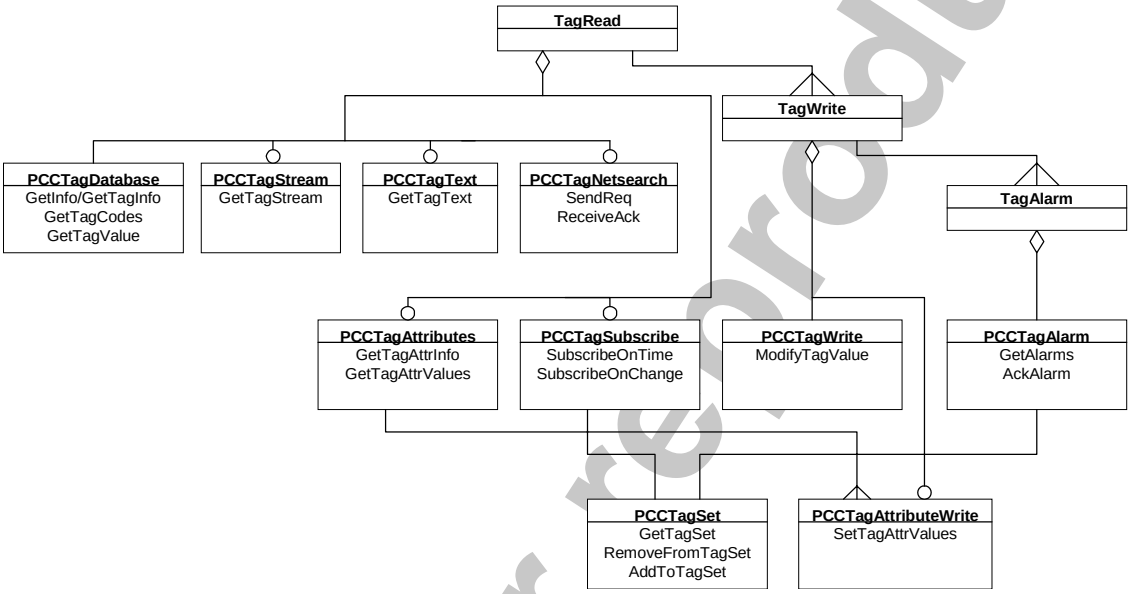

Figure 17 - Tag interface components relationships

The diagram represents each of the basic application types as a specialisation of its super-class where each specialisation level aggregates more component interfaces. The empty rings denote optional component interfaces and the derivation triangle a class that needs its super-class for instantiation. The attributes are the connection points.

D.6 File structure

In addition to the classes described in the previous diagrams, there is also a general data type definition file and a file with standard tag names. The below table defines the files and their contents.

Table 5 - Tag related companion standards

Companion standard	Description	File name
PACReadableServer	Application for reading	tagread.mcs
PACWritableServer	Application: Add write	tagwrite.mcs
PACAlarmSystem	Application: Add alarm handling	tagalarm.mcs
TagData	Data type definitions	tagdata.mcs
PCCTagDatabase	General data base functionality	datag.mcs
PCCTagText	Read text value tags	datat.mcs
PCCTagStream	Read a stream address	datas.mcs
PCCTagNetsearch	Search network for tags	datan.mcs
PCCTagAttributes	Read and find attributes	dataatt.mcs
PCCTagSubscribe	Subscribe to tag values	datasub.mcs

Companion standard	Description	File name
PCCTagWrite	Write tag values	dataw.mcs
PCCTagAlarm	Read and handle alarms	dataa.mcs
PCCTagSet	Define set of tags	dataset.mcs
PCCTagAttributeWrite	Write attribute values	dataattw.mcs
TagStandard	Standard tag names	tags.mcs

D.7 Standard tag names

D.7.1 General

D.7.1.1 Internal and external representation

Note that the tag name has to be encoded in a fixed number of characters and, due to protocol requirements, adhere to a fixed structure. These rules apply only to the protocol and internal representation may use other formats, e.g., more compact to save storage or search times. It may also be useful to format the tag name differently for presentation to humans, although this may cause problems with recognition of the same tag name on different systems.

D.7.1.2 Tag name length

The tag name is limited to 24 characters maximum. Shorter tag names shall be null terminated.

D.7.1.3 Character set

All standard tag names (P-type and S-type) shall only use upper case letters (‘A’ to ‘Z’ inclusive), lower case letters (‘a’ to ‘z’ inclusive) or decimal numbers (‘0’ to ‘9’ inclusive). In addition, the special character dash (‘-‘), under score (‘_’) or period (‘.’) can be used.

Character in this context is the basic PISCES type `char8_m`.

D.7.1.4 General tag name structure

All tag names shall have a structure as described below:

a) **Tag name class**: The first character shall be a lower case letter identifying the tag name group. Currently the following groups are defined:

   p: Tag with name conformant to the PCS rules presented here

   y: Yard tag with name structure defined by external entity.

   s: Standard tag pointing to a ship independent information item

   i: Internal tag defined by manufacturer.

b) **Tag name body**: The rest of the name is structured dependent on the tag name class.

D.7.2 Structure of P tag name class

D.7.2.1 Introduction

The P tag name class body is structured according to the rules presented in this clause. The name body will be divided into groups, each consisting of a defined number of upper case characters followed by from zero to any number of decimal numbers. The name is structured so that it can be parsed by a regular expression.

D.7.3 General structural rules

D.7.3.1 Outline structure

The outline format for the p class tag name is presented below.

```
pMGnn.SGnn.TC.nn
p                       - Identify p class.
 MG                     - Main group (two letters)
   nn                   - Optional main group instance number
      SG                - Sub group (two letters)
        nn              - Optional sub group instance number
           TC           - Data type code (two letters)
              nn        - Unique serial number
```

Each group of the tag is delimited by a period (full stop), except after the first 'p'. The main group and the sub group may have an instance number immediately following.

D.7.3.2 Tag name length and encoding

The maximum length of 24 characters allows group and sub-groups of up to three digits and a serial number of up to eight digits. It is possible to compress this in an internal representation by omitting dots and the leading 'p'.

Special coding with more than three digits group numbers can be used for certain tag types, e.g., container or other modular cargo. However, the total length shall not exceed maximum name length.

D.7.3.3 Group and sub-group number structure

The group and sub group number shall be a decimal number, without leading zeros. In cases where there are only one instance of the indicated group on board (e.g., only one main engine), the instance number shall be omitted.

D.7.3.4 Serial number structure

The serial number will normally be a manufacturer dependent serial number intended to distinguish between otherwise identical tag names. For some types of tags (e.g., contain related identifiers), the serial number may contain structural information.

D.7.3.5 Uniqueness of name

The tag name must be unique within a MAU. It should be unique over the ship (the PISCES network), although this is more difficult to ensure.

The main group codes must be unique. The general sub-group codes are unique among sub-groups (achieved by assigning special first letters to these groups).

Other sub-group codes must be unique among the main groups in which it is used.

D.7.4 Main process codes

The main process code consists of two upper case letters optionally followed by a decimal number. The below table lists the currently defined codes.

Table 6 - Main process codes

Process code	Number	Explanation
MP	Engine	Propulsion Engines
MG	Engine	Generator and auxiliary engines
ML		Lubrication oil systems
MC		Cooling Systems, fresh and/or salt water
MB		Boiler
MD	Shaft	Drive Train, i.e, shafts, gears, clutches, propellers
MF		Fuel Oil systems

MM		Miscellaneous Machinery
CB	Tank	Ballast system
CL	Tank	Liquid Cargo Tanks
CH	Hold	Bulk Cargo
CR		Reefer related entities, cooling (except CH groups)
CM	Deck	Modular cargo on decks (e.g., RO-RO)
CC	Hold	Container Cargo
CO		Other/General Cargo
SH		Ship data (name, captain, yard)
HU		Hull related data
HF		HVAC, Climate control, Provisions, Waste, Sanitary
NA		Navigation (position, speed, ARPA, ECDIS etc.)
EV		Environment (wind, waves, weather)
FA	Central	Fire and gas alarm
SY		System/Sub-system (monitoring and alarm system itself)
OT		Other/Miscellaneous

The number column specifies what the number code, if used, shall indicate. The number field shall not be used if there is only one instance of the device (e.g., main machinery) on board.

Note that machinery and cargo and ballast main-groups form two super-groups. These super-groups use the same first character ('M' and 'C' respectively).

D.7.5 Process sub-codes

The second group consists of two upper case letters that defines a sub-group for the main process group. The sub-groups are divided into three classes dependent on whether they are used anywhere on the ship (general sub-groups), whether they are used within one super-group (e.g., machinery or cargo) or if they are specific to one single main group. The sub-group code can be followed by a number as for the main code.

D.7.6 General sub-groups

The following table contains the currently defined sub-groups that are in general use over more than one main process group. All codes use 'X', 'Y' or 'Z' as first character. These characters are reserved for these groups.

Table 7 - General sub-groups

Sub-group	Number	Explanation
ZZ		No specific subgroup
XP		Pump
XV		Valve
XE		Electrical motor
XT		Tank
XM		Manifold
XL		Pipe-line, tube
XC		Compressor
XS		System/Subsystem (network, monitoring system itself)
XH		Heat exchanger

Numbering will normally be dependent on the main group in use.

D.7.7 Automation related sub-group

This standard does not cover general automation and no more details in this area is provided here.

D.7.8 Navigation sub-groups

The following table identifies navigation related sub-groups.

Table 8 - Navigation sub-groups

Sub-group	Number	Explanation	Main Process Codes
GP		GNS receiver	NA
LC		Loran C/Chaicka receiver	NA
AR		ARPA Radar	NA
EC		ECDIS/ECS	NA

D.7.9 Data type indication group

The third group is two upper case letters specifying what kind of information item this is. The code is based on a simplified version of general process equipment coding rules.

Table 9 - Data type indicators

Letter	First position meaning	EU	Second position meaning
A	Angle	rad	Alarm (no indication - "binary")
B			
C	Conductivity (Electrical)	Ohm or S	Control (output)
D	Density/Specific Gravity	kg/m^3	Documentation (data models, text: in)
E	Voltage	V	
F	Flow	m^3/s	
G	Dimensions	m	
H			
I	Current (electrical)	A	Indication (input)
J	Power	kW	
K	Time	s	
L	Level	m	HMI related data (input)
M	Moisture or humidity	%	Maintenance/calibration data/history (in)
N			
O			
P	Pressure	Bar	Parameter (filter, trend: in or out)
Q	Quantity, event or counter		
R			Record/trend (input)
S	Speed or frequency	Hz, m/s, knots, RPM	System status codes (in or out)
T	Temperature	^oC	
U	Function block (composite)		Multifunction (in or out)
V	Viscosity		Version/revision codes (input)
W	Weight or force	kg or N	

X	Any meaning		Any meaning
Y	System level		
Z	Position	m or nm	

The first character defines the type of data entity pointed to by the tag. Of special interest are:

- **U**: This code is used for composite data entities (function blocks).
- **X**: This code is used for entities not otherwise defined.
- **Y**: This code is used for entities relating to monitoring and alarm device itself.

The second character specifies if the entity is an output or input and if it is related directly to a physical state (alarm, indication or control) or if it is related to more system oriented information (HMI, documentation, version codes etc.).

A complex function block with several inputs and/or outputs would normally be coded as '**UX**'.

D.7.10 Use of engineering units

The engineering unit in use will be available for the general alarm and monitoring system by looking up static attributes of the tag. However, as a rule, the SI units corresponding to the indicated measurement type (first character) shall be used.

The preferred engineering unit is listed in the EU column.

D.7.11 Sequence number

The last part of the tag name is a sequence number code. This code is specific to a particular manufacturer or system integrator and cannot in general be relied on to have any specific meaning. The sequence number shall consist of decimal digits only. Leading zeros are allowed.

D.8 Structure of standard tags (S class)

The standard tags (s name class) will use the same format as the p class tags, except that the leading letter will be a lower case 's'.

The ‘s’ will show that this tag is a ship independent measurement with properties defined in a general ship operational data model. The prparation of this model is out of the scope of this standard.

D.9 Structure of yard tags (Y class)

The yard tag structuring shall be determined by the yard or any other authority that has an overall responsibility for the design of the ship. The main purpose of the yard tags is to provide a link between the automation system and the physical ship. It is suggested that all indicators identified by a yard tag shall have the same yard tag on their corresponding measurement or alarm.

D.10 Structure of internal tags (I class)

Internal tags have no particular rules for structuring other than the first character being the lower case letter ‘i’.

D.11 New tag name classes

Other tag name classes can be defined by the standard organisations that maintain these specification documents. No user should rely on any specific leading letter being free for own internal use.

D.12 General quality indicators

All information retrieved through the tag database mechanisms will be quality controlled by the providing system. This quality control consists of several parts as discussed in the following clauses.

D.13 Certification

The quality control principles for a specific application may have been checked and certified by some agency or by the manufacturer himself. This certification is not mandatory and it is considered out of scope of this standard. However, the `GetInfo` MCP provides a possibility to specify if any certificate has been awarded to the quality control mechanisms. The actual certificate or specification document must be obtained from the manufacturer.

D.14 Time stamp

All measurements shall be marked with the time at which they were collected. For raw sensor data, the time stamp should be the time of the data acquisition. For derived data, it may be an estimate of the time of validity.

The time stamp is accurate within the limits defined by the `sampleTime` attribute of the relevant `TagInfo` data block.

D.15 Validity flag

All measurements shall be marked with a validity flag saying if the value has the required quality or not. The flag can also attempt to quantify the level of non-conformance with quality requirements. Legal values for the validity flag are defined as the interpretation `StateCode`.

The required quality is defined in the `TagInfo` data block in the form of the `precision` and `sampleTime` attributes.

D.16 Authentication

The application providing this interface shall specify to what level the data values are authenticated, i.e., guaranteed to be not tampered with. Normally, this will depend on the mechanisms for data acquisition used by the application.

The authentication level is defined by the `authentication` flag in the `TagInfo` data block.

D.17 Companion standard specifications

D.17.1 DATA TYPES TagData

```
DATA TYPES TagData
  * This specification contains general data definitions for tag based
    data access.

  * Revision History
    990831  1.1 For IEC
    990512  1  Longer tag name, tag info, removed setin/out, added unit
               error codes.
    981130  A  First, based on old interface CTagDatabase. Added some
               more on authentication.

  VERSION     1.1
  DATE        1999-08-31
  RESPONSIBLE IEC TC80/WG6

;-------------------------------------------------------------
REFERENCES
```

```
  General

;-------------------------------------------------------------
GLOBAL

  ;-------------------------------------------------------------
  ; Tag identities
  ;-------------------------------------------------------------
  INTERPRETATION TagNumber OF int32_m
    * The container for tag numbers. The number zero is reserved as no
      tag.

    0 = NO_TAG            ; Undefined tag

  ;-------------------------------------------------------------
  INTERPRETATION TagName OF [32]char8_m
    * Tag name for a measurement point, with terminating null or
      termination at end of array (max 32 significant characters).

    * Some interfaces will contain both internal tag codes constructed
      as specified in this standard and yard specific codes. This
      means that different tag names can have the same tag number.

  ;-------------------------------------------------------------
  DATA BLOCK TagId
    * An aggregate of tag name and number. Note that different tag
      names may have the same number (aliasing of name is allowed).

    TagNumber   number     ; Tag number
    TagName     name       ; Tag name

  ;-------------------------------------------------------------
  INTERPRETATION TagSet OF int32_m
    * An identifier for a set of tags. A valid set id points to an
      internal structure in the server that lists all tags in the set.
      Some servers can have statically defined sets. The value zero is
      used to indicate no valid set. The ALL_TAGS set defines all tags
      in the server (not always supported as set).

    0 = NO_SET
    1 = ALL_TAGS

  ;-------------------------------------------------------------
  ; Various application classification info
  ;-------------------------------------------------------------
  INTERPRETATION Conformance OF [32]bool_m
    * These bit fields are used to indicate conformance level for this
      interface. This corresponds to the number of components supported
      with the TagData interface taken as implicit. Additional
      components are:

    * The INCMAP/DECMAP flags indicate if the number
      of tag codes in use changes during the life time of the
      server. The first means that the number of tags may increase,
      the second that it may decrease.

    0 = TC_INCMAP         ; Increment tag number map
    1 = TC_DECMAP         ; Decrement tag number map
      ;
    2 = TC_ATTRIBUTES     ; Additional support of TagAttributes
    3 = TC_ALARM          ; Additional support of TagAlarm
    4 = TC_WRITE          ; Additional support of TagWrite
    5 = TC_COMPLEX        ; Additional support of TagDataComplex
    6 = TC_STREAM         ; Additional support of TagDataStream
    7 = TC_SUBSCRIBE      ; Additional support of TagDataSubscribe

  ;-------------------------------------------------------------
  INTERPRETATION Certificates OF [16]bool_m
    * These bit fields are used to show which certificates the
      interface has. All false means no certificates.

    0 = TC_OWNQA         ; Own documentation on data QA
    1 = TC_EXTQA         ; External certificate for data QA
    2 = TC_OWNDA         ; Own documentation on authentication
    3 = TC_EXTDA         ; External certificate for authentication
```

```
;-------------------------------------------------------------
INTERPRETATION TagKeyCode OF [32]bool_m
  * These tag keys are used to specify the allowed key type searches
    supported by an implementation of the interface. The key is one
    bit in a 32 bit word (max is 24 bits as 8 is allocated to
    implementation specific keys).

  0 = KEY_NONE          ; No key based search, returns all tag codes
  1 = KEY_NUMBER        ; Search on tags by tag number allowed
  2 = KEY_NAME          ; Search on tags by name allowed
  3 = KEY_WCNAME        ; Wildcard search by name allowed
  4 = KEY_LOC           ; Search by location code (LocationCode)
  5 = KEY_TYPE          ; Search on tag type (TagType)
  ;
  23 = KEY_SETDEF       ; Can/will define a tag set
  24 = KEY_USER         ; Implementation specific keys to 31

;-------------------------------------------------------------
INTERPRETATION TagKey OF char8_m
  Contains char8_m key search pattern. The string shall be null
  terminated or terminated at end of array. It is legal to send an
  empty string for KEY_NONE. Only one search bit can be set from the
  following possibilities:

  * KEY_NONE: No pattern (length = 0), Return all tags.

  * KEY_NUMBER: A single number, a series of decimal numbers
    separated by comma (,), and/or a range shown by two numbers
    separated by a minus sign (-), e.g., "23", "23,24,26" or
    "20-25". In the latter case, the empty string represents
    infinity, e.g., "0-" is all numbers. The numbers represent tag
    code numbers. This function is most useful to define sets.

  * KEY_NAME: A single tag name or a series of tag names separated
    by comma: "tag1" or "tag1,tag2"

  * KEY_WCNAME: One name with the following wildcards:
     - `?' (question mark) - any one legal character,
     - `*' (asterix) - any length sequence of any legal character
     Examples: "tag?" and "ta*" both match "tag1" and "tag2", ta* is
     the only that match "tag10".

  * KEY_LOC, KEY_TYPE: A number (as KEY_NUMBER), with location codes
    (see general data types) instead of tag code numbers.

;-------------------------------------------------------------
INTERPRETATION TagType OF int16_m
  * This code is used to specify the type of a tag.

  0 = TT_VALUE          ; Plain numeric type (f64)
  1 = TT_TEXT           ; Plain text type ([64]c8)
  2 = TT_ALARM          ; Alarm type, VALUE with alarm limits
  3 = TT_STREAM         ; Stream/file type
  4 = TT_FB             ; Function block, value with attributes
  5 = TT_COMPLEX        ; Other formatted type

;-------------------------------------------------------------
INTERPRETATION TagAuthentication OF int16_m
  * This code specifies the mechanism used to authenticate a given
    tag. This applies to both originator and quality. TQ_NOTAMPER
    and TQ_AUTH is used if the device flags all internal and external
    precision loss and tampering or if the value cannot be tampered with
    or lose precision. Other flags are set to qualify this
    statement. Lower values means better authentication.

  0 = TQ_NOTAMPER  ; Value cannot be tampered with, quality controlled
  1 = TQ_AUTH      ; Data is fully authenticated, all exceptions flagged
  10 = TQ_CAUTH    ; Full authenticated internally and controlled source
  20 = TQ_USOURCE   ; Full authenticated internally uncontrolled source
  30 = TQ_TOPERATOR ; Operators are trusted and checked, but changes
                    ; are not flagged.
  ;
  1000 = TQ_NONE    ; No authentication in force
```

```
;-------------------------------------------------------------
INTERPRETATION TagSemantics OF [16]bool_m
  * This code is used to specify the semantics of a tag. Read and
    write flags are not exclusive: Both means that the system can
    perform a function call with output dependent on input. Constant
    and unfiltered can be used in conjunction with other flags
    (normally only read).

  0 = TS_UNKNOWN
  1 = TS_READTHROUGH     ; Read directly from physical unit
  2 = TS_READBUFFER      ; Read from physical unit via buffer
  3 = TS_WRITETHROUGH    ; Write directly to physical unit
  4 = TS_WRITEBUFFER     ; Write to physical unit via buffer
  5 = TS_CONSTANT        ; Constant value
  6 = TS_UNFILTERED      ; No anti-aliasing done
  7 = TS_QUEUED          ; Events are queued, no changes lost

;-------------------------------------------------------------
DATA BLOCK TagInfo
  * Static information about a tag.

  * If engineering unit is undefined (EU_OTHER), the text
    representation of the unit (for printing purposes) shall be
    defined.

  * The precision is the minimum scalar distance that is
    necessary to say that a difference between two measurements is
    significant. This has meaning for scalar measurements and
    usually also for other multi-dimensional measurements, e.g.,
    position (typically use length of distance vector).

  * The sample interval is the maximum time before the server has a
    new measurement of the tag value ready. Check semantics for
    meaning of sample time (read and write through renders it
    meaningless). Note that tag values normally are anti-aliasing
    filtered before being supplied to user (based on sample
    time). The unfiltered semantics flag means that this is not
    done.

  * The tagAttributes entry specifies the number of
    extra attributes that the tag data base stores (see
    TagAttributes type for predefined attributes).

  * The fieldFlags shall be set for all valid entities. All false
    means that the block is invalid. All true is legal if non-used
    entities contain a proper value (including null terminated
    strings and null values).

  TagNumber         tagNumber      ; Numeric code for tag
  [48]char8_m       description    ; Description of tag
  TagType           tagType        ; Type of access mechanism
  EngineeringUnit   engUnit        ; Engineering unit or equivalent
  [8]char8_m        euText         ; Textual engineering unit
  TagSemantics      tagSemantics   ; Semantics of tag operation
  TagAuthentication authentication ; Quality control mechanisms
  float64_m         precision      ; Measurement precision
  word32_m          sampleTime     ; Sample interval in ms
  int16_m           tagAttributes  ; Attributes supported
  [10]FieldOk       fieldFlags     ; Field validity flags

;-------------------------------------------------------------
; Dynamic values
;-------------------------------------------------------------
INTERPRETATION StateCode OF int16_m
  * State codes. Unused codes are user defined. For future expansion
    these codes should start at USERSTATES. Note the classifications
    of state codes. they are in increasing value:
    - SC_NORMAL is normal
    - Up to and including SC_AUTHENT means that value may have lost
      authenticity, but is still under control.
    - Up to a.i. SC_PRECISION loss in precision
    - Up to a.i. SC_UNRELIABLE possible spurious or locked value
    - Up to a.i. SC_DEFECT sensor malfunction
    - Up to a.i. SC_OPERR  operation errors
```

```
    0 = SC_NORMAL        ; operation completed as expected
    1 = SC_FILTERED      ; Filter may affect value "unreasonably"
    2 = SC_OPERATOR      ; Operator set value
    5 = SC_AUTHENT       ; Other event that can effect authenticity
      ;
    6 = SC_PRECISION1    ; Loss in precision by factor 10
    7 = SC_PRECISION2    ; Loss in precision by factor 100
    8 = SC_PRECISION3    ; Loss in precision by factor 1000
    9 = SC_PRECISION     ; Unknown loss in precision
      ;
    10 = SC_TIMEOUT      ; Update timeout exceeded by source
    19 = SC_UNRELIABLE   ; Unreliable value
      ;
    100 = SC_BADSENSOR   ; Sensor specific errors follows
    101 = SC_OPEN        ; Open circuit
    102 = SC_CLOSED      ; Closed circuit
    103 = SC_SHORT       ; Short circuit
    104 = SC_BROKEN      ; Broken connection
    105 = SC_NOT_AVAILABLE ; Input or output is not available
    106 = SC_MAINTENANCE ; Unit under maintenance
    107 = SC_BLOCKED     ; Input or output is blocked by operator
    199 = SC_DEFECT      ; Unit is defect
        ;
    300 = SC_NOOP        ; Specified tag does not support operation
    301 = SC_NOTAG       ; No such tag
    302 = SC_NOSET       ; No such tag set
    399 = SC_OPERR       ; Unspecified error in data retrieval
      ;
    400 = SC_UDEAD       ; Unit itself is dead
    401 = SC_ULINK       ; Link to unit is dead
    499 = SC_UCODES      ; Last unit related error
      ;
    10000 = SC_USERSTATES

  ;-------------------------------------------------------------
  INTERPRETATION AlarmState OF [16]bool_m
    * Alarm states bit map. All false is normal. Use alarm interface
      to retrieve detailed alarm state information. NONESSENTIAL
      shall only be set if there is another bit set and if the signal
      originates from a system that is not defined as essential, i.e.,
      that alarms shall have a lower priority than for essential
      systems. WARNING need not be set when ALARM is set.

    0 = AC_WARNING       ; Value is outside normal, but no alarm
    1 = AC_ALARM         ; Value is in alarm area
    2 = AC_NA_WARNING    ; Non-acknowledged warning(s) exists
    3 = AC_NA_ALARM      ; Non-acknowledged alarm(s) exists
    4 = AC_NONESSENTIAL  ; This signal is not from an essential system

  ;-------------------------------------------------------------
  DATA BLOCK TagValue
    Values for one tag with standard f64 format.

    TagNumber       tagNumber   ; Numeric code for tag
    StateCode       state       ; State code
    AlarmState      alarm       ; Most important active alarm
    GlobalTime      time        ; Last updated
    float64_m       value       ; Current value

  ;-------------------------------------------------------------
  DATA BLOCK TagText
    Values for one tag with standard character format

    TagNumber       tagNumber   ; Numeric code for tag
    StateCode       state       ; State code
    GlobalTime      time        ; Last updated
    [64]char8_m     value       ; Current value

  ;-------------------------------------------------------------
  ; Attribute related data
  ;-------------------------------------------------------------
  INTERPRETATION TagAttrNumber OF int32_m
    * Each tag can have certain attributes associated with it.
      Normally, these are alarm limits and sometimes filter constants.
      This list specifies some common attributes. Additional
```

```
    attributes can be defined for an interface by codes from TA_USER
    and higher.

    0 = TA_NONE
    1 = TA_ALARMLOW
    2 = TA_ALARMHIGH
    4 = TA_ALARMLOWLOW
    3 = TA_ALARMHIGHHIGH
    ;
    1000 = TA_USER

;-------------------------------------------------------------
INTERPRETATION TagAttrStatus OF int16_m
  * The status of an attribute value can be:

   0 = TAS_VALID        ; value is valid and activated
   1 = TAS_DISABLED     ; value is valid, but disabled
   2 = TAS_NOATTRIBUTE  ; no such attribute for tag
   3 = TAS_NOTAG        ; no such tag

;-------------------------------------------------------------
DATA BLOCK TagAttrInfo
  * This is the container for an attribute information structure.
    Text strings are empty if not used. The description string
    should be suitable for printing out information about the
    attribute in a table, e.g., for alarm limits.

  TagAttrNumber   attribute   ; Attribute code
  [32]char8_m     description ; Description of attribute
  EngineeringUnit engUnit     ; Engineering unit or eqivalent
  [8]char8_m      euText      ; Textual engineering unit
  BlockOk         valid       ; valid flag

;-------------------------------------------------------------
DATA BLOCK TagAttrValue
  * This is the container for an attribute

  TagAttrNumber   attribute   ; Attribute code
  TagNumber       tagNumber   ; Tag number
  TagAttrStatus   status      ; Status of value
  float64_m       value       ; The value

;-------------------------------------------------------------
DATA BLOCK TagAttrValueW
  * This is the container for an attribute write value

  TagAttrNumber   attribute   ; Attribute code
  TagNumber       tagNumber   ; Tag number
  float64_m       value       ; The value

;-------------------------------------------------------------
; Alarm related data definitions
;-------------------------------------------------------------
INTERPRETATION AlarmSequence OF int16_m
  * A tuple of tag number and alarm sequence number will identify an
    alarm instance. This is the sequence number.

;-------------------------------------------------------------
INTERPRETATION AlarmCode OF int32_m
  * Alarm codes. Zero is normal. Codes above USERALARMS are user
    defined. Codes below USERALARMS are reserved for future
    expansion.

  0 = AC_NORMAL
  1 = AC_SOMEALARM         ; Undefined type of alarm
  2 = AC_LOWLOW            ; Signal has very low value
  3 = AC_HIGHHIGH          ; Signal has very high value
  4 = AC_LOW               ; Signal has low value
  5 = AC_HIGH              ; Signal has high value
  6 = AC_INSTRUMENT_HIGH   ; Input value too high for instrument
  7 = AC_INSTRUMENT_LOW    ; Input value too low for instrument
  8 = AC_DEVIATION_LOW     ; Low deviation between signals
  9 = AC_DEVIATION_HIGH    ; High deviation between signals
  10 = AC_RATE_OF_RISE     ; Signal rising too fast
  11 = AC_OSCILLATIONS     ; Signal oscillating
```

```
    12 = AC_TOO_LONG         ; Too slow response
    13 = AC_OPEN             ; Input open
    14 = AC_CLOSE            ; Input closed
    15 = AC_ERROR            ; Internal instrument error
    16 = AC_OPERATOR         ; Operator intervention in instrument
    ;
    100000 = AC_USERALARMS

  ;-------------------------------------------------------------
  DATA BLOCK TagAlarmValue
   * Values for one tag with alarm information. The value will have to
     be interpreted according to the information specified in the
     TagInfo data block. Use state code to check if block is ok.

    TagNumber       tagNumber   ; Numeric code for tag
    AlarmSequence   seqNumber   ; Alarm instance for this tag
    StateCode       state       ; State code
    AlarmCode       alarm       ; Alarm state code
    AlarmState      aState      ; Importance of alarm
    GlobalTime      time        ; Time of trigger
    float64_m       value       ; Current value
    float64_m       limit       ; Limit that were broken
```

D.17.2 Application PACReadableServer

```
APPLICATION PACReadableServer DERIVED FROM PACFullApplication

  * This application contains the general framework for the creation
    of an interface to any type of data server. Based on a tag name
   (16 character text string), it is possible to read the data item.

  * Revision history :
    1999-08-31 1.0, ojr, For IEC
    1998-11-30 A, ojr : First

  VERSION     1.0
  DATE        1999-08-31
  RESPONSIBLE IEC TC80/WG6

;-------------------------------------------------------------
REFERENCES

 PCCTagDatabase

  ; The following interfaces are optional, and can be added

; PCCTagText
; PCCTagSubscribe
; PCCTagAttributes
; PCCTagNetsearch

;-------------------------------------------------------------
USAGE

  * The application is the minimum implementation of a tag based data
    base reader. Refer to the individual interface specifications for
    detailed discussion of functionality. Additional interface
    components can be added to support subscription or network wide
    search capabilities. This has to be done as derivations from this
    class.

;-------------------------------------------------------------
INTERFACES

  ACCEPT TagData

    INTERFACE COMPONENT PCCTagDatabase

    ; And optionally one or more

;   INTERFACE COMPONENT PCCTagText
;   INTERFACE COMPONENT PCCTagSubscribe
;   INTERFACE COMPONENT PCCTagAttributes
```

```
  ; If net search shall be used, one must also provide accept and connect
  ; interfaces.

; CONNECT ABCM1

;   INTERFACE COMPONENT PCCTagNetsearch

; ACCEPT ABCM1

;   INTERFACE COMPONENT PCCTagNetsearch
```

D.17.3 Application PACWritableServer

```
APPLICATION PACWritableServer DERIVED FROM PACReadableServer

  * This application contains the general framework for the creation
    of an interface to any type of data server. Based on a tag name
   (16 character text string), it is possible to read and write the
    data item.

  * Revision history :
    1999-08-31 1.9, ojr, IEC issue
    1998-11-30 A, ojr : First

  VERSION     1.0
  DATE        1999-08-31
  RESPONSIBLE IEC TC80/WG6


;-------------------------------------------------------------
REFERENCES

 PCCUserAuth
 PCCTagWrite

  ; The following interfaces are optional, and can be added

; PCCTagAttributeWrite

;-------------------------------------------------------------
USAGE

  * The application is the minimum implementation of a tag based data
    base writer. Refer to the individual interface specifications for
    detailed discussion of functionality. Additional interface
    components can be added to support attribute write.

;-------------------------------------------------------------
INTERFACES

  ACCEPT Authenticate
    * Need one interface for user authentication.

    INTERFACE COMPONENT PCCUserAuth

  ACCEPT TagData
    * and the actual write interafce.

    INTERFACE COMPONENT PCCTagWrite

    ; And optionally

;   INTERFACE COMPONENT PCCTagAttributeWrite
```

D.17.4 Application PACAlarmSystem

```
APPLICATION PACAlarmSystem DERIVED FROM PACWritableServer

  * This application contains the general framework for the creation
    of an interface to any type of data server. Based on a tag name
   (16 character text string), it is possible to read and write the
    data item.

  * Revision history :
```

```
    1999-08-31, 1.0, ojr, IEC issue
    1998-11-30 A, ojr : First

  VERSION     1.0
  DATE        1999-08-31
  RESPONSIBLE IEC TC80/WG6

;-------------------------------------------------------------
REFERENCES

 PCCTagAlarm

;-------------------------------------------------------------
USAGE

  * The application is the minimum implementation of a tag based alarm
    system. Refer to the individual interface specifications for
    detailed discussion of functionality.

;-------------------------------------------------------------
INTERFACES

  ACCEPT TagData

    INTERFACE COMPONENT PCCTagAlarm
```

D.17.5 Interface PCCTagDatabase

```
INTERFACE PCCTagDatabase
  * This interface contains the basic functionality for reading and
    writing tag based data items from a data base. This part of the
    interface is used to access the tags data base and determine
    properties of tags.

  * Revision History
    990831  1.0 IEC issue
    990512  F  Unit status, length fields
    981201  E  Removed data def
    980810  D  Separate text interface, name removed from info and
               into id. Use just alarm classes. Added set and various
                other.
    980810  C  Changed class specification
    971202  B  Removed tag access to TagDataValue, Added info
    971111  A  First release, based on interface "Data"

  VERSION     1.0
  DATE        1999-08-31
  RESPONSIBLE IEC TC80/WG6


USAGE
  * This interface allows the look-up of tags to get interface specific
    numeric tag codes. Tag codes can be retrieved from the GetTagCodes
    entry based on a key search. Several key types may be legal. It is
    possible to retrieve information on the tags and the current
    value. Notes to implementors:

    * This interface uses an internal code (TagNumber) to reference a
      tag. The value of this code for a given tag may or may not be
      the same between two different connects to the MAU implementing
      the server. The GetInfo instance code should be used after each
      server restart to check if tag code mappings has been changed.
      This code shall reflect the following:

      * It is legal to build the tag data base incrementally while the
        server is running.

      * It is legal to remove tags from the data base while the server
        is running.

      * It is not legal to reuse internal codes for different tags.

    * The tag data server may allow the client to search for tags
      based on various keys (TagKeyCode). However, some servers may
```

```
      have constant tag code mappings and no search option at all.

    * All return blocks are less than 2000 bytes long.

;-------------------------------------------------------------
REQUIRED DOCUMENTATION

  TAGLIST    ; List of tag names supported by the interface. The list
             ; should preferably contain engineering units etc.

;-------------------------------------------------------------
REFERENCES
  General
  TagData

;-------------------------------------------------------------
CONNECTION POINTS

  ;-----------------------------------------------------------
  SUBSCRIBE GetInfo
    * Used to retrieve information about interface. Subscribable,
      changes in configuration (number of tags) will be reported to
      all subscribing clients. Note that mapping cannot change during
      connection time as reuse of tag codes are not allowed.

    * The first fields are the number of tags supported by the
      interface and what search keys it supports (see GetTagCodes).

    * The instance code can be used to check if the configuration of
      the server MAU has changed from the last invocation. It shall be
      incremented each time the _mapping_ between tag codes and tag
      names has been changed (i.e., constant code shows that the
      mapping is constant). Note that increment or decrement in tag
      number do not imply that the mapping has changed. The value zero
      means that the mapping changes each time the server MAU
      restarts. Reuse of tag codes is not allowed during server MAU
      life-time.

    * The conformance flags defines what additional extensions to the
      interface that are available. The inc/dec flags may or may not
      cause the instanceNo value to be zero (i.e., it is possible to
      have changing number of tags where the mapping between each tag
      and tag code is kept constant => instanceNo is constant
      non-zero).

    * The certificates field specify if the device has been certified
      with respect to data authentication and data quality control. The
      user need to check the manufacturer and equipment type to get
      hold of the relevant certificates.

    * The unit state codes indicate state of the physical unit generating
      data. Errors in this (other values then SC_NORMAL – zero) means that
      all tags are stuck at last value. No further errors will be generated.

    OUTPUT
      word32_m     noOfTags       ; Number of tags in interface
      TagKeyCode   keys           ; Search keys supported
      word32_m     instanceNo     ; instance/version code
      Conformance  conformance    ; Conformance level
      Certificates certified      ; QA certificates flags
      StateCode    unitState      ; state of physical unit
      BlockOk      ok             ; true if data block is valid

    * Precondition
      none

    * Postcondition
      returns information. unitState will indicate if the interface can be
      used or not.

  ;-----------------------------------------------------------
  FUNCTION GetTagCodes
    * Used to retrieve numeric tag codes for specified search pattern.
      The first two input numbers are used to continue upload. Some
      interface instances may have this function as a dummy, in which
```

```
      case it always returns NOT_IMPLEMENTED.

    INPUT
      int32_m                startIndex  ; Start returning records here
      TagKeyCode             keyType     ; Type of key used
      [word16_m:1500]TagKey  key         ; Search key and/or define set

    OUTPUT
      word8_m     status        ; Request status
      bool_m      more          ; More hits
      int32_m     endIndex      ; The hit index of the last code
      TagSet      setCode       ; The set code if set requested
      [word16_m:48]TagId  ids   ; Returned tags

    * Precondition

      * keyType must be one of the legal key types for this interface.

      * The startIndex entry shall be zero for first call on new
        search. To get more entries than can be returned by one call,
        startIndex shall be set to the previously returned endIndex
        and key kept constant for following calls.

      * Note that tag names can be aliased and that the same number
        may appear several times in different named ids.

    * Postcondition

     * status is zero for everything all right other error codes for
       status are:
      - BAD_KEY (= 1), Illegal key type (more than one key or unsupported
        key).
      - BAD_STRING (=2), Search key could not be interpreted (errors).
      - NOT_IMPLEMENTED (=3), function not implemented.
      - SET_NOT_SUPPORTED (=4), returns valid codes, but defines no
       set.

      * more is true if there may be more hits. endIndex specifies the
        internal index of the next tag to be searched. Note that
        startIndex and endIndex is used to point into the server's
        internal data base and may not have any external
        interpretation.

      * Note: Search on one tag number shall result in more than one hit if
        several tag names (e.g., one yard tag, one internal tag and one
        p-tag) is mapped to the same tag code.

  ;-------------------------------------------------------------
  FUNCTION GetTagInfo
    * Used to retrieve a number of tag information entries.

    INPUT
      [word16_m:22]TagNumber  tagCode   ; Code numbers

    OUTPUT
      [word16_m:22]TagInfo    info      ; Returned info

    * Precondition
      none

    * Postcondition
      Returns the number of tags that were *found*. This may be less
      than that requested, if some requested codes are undefined.
      Check numbers to be sure of mapping. Non-returned numbers mean
      that the tag number does not exist.

  ;-------------------------------------------------------------
  FUNCTION GetTagValue
    * Used to retrieve a number of values using an array of tag codes.
      This can be used on all standard tags that use f64 format.

    INPUT
      [word16_m:82]TagNumber   tagCodes

    OUTPUT
```

```
      [word16_m:82]TagValue    values

    * Precondition
      none

    * Postcondition
      Return all tags found (may be less than requested if some
      requested codes are undefined). Check state code to verify
      validity of values and numbers to check existence of tags.
```

D.17.6 Interface PCCTagText

```
INTERFACE PCCTagText
  * This interface contains additional functionality to PCCTagDatabase
    to read text strings.

  * Revision History
    990831  1.0 IEC issue
    981201  A  Added text

  VERSION     1.0
  DATE        1999-08-31
  RESPONSIBLE IEC TC80/WG6

USAGE
  * Use together with PCCTagDatabase

  * All return blocks are less than 2000 bytes long.

;-------------------------------------------------------------
REQUIRED DOCUMENTATION

  TAGLIST    ; List of tag names supported by the interface. The list
             ; should preferably contain engineering units etc.

;-------------------------------------------------------------
REFERENCES
  General
  TagData

;-------------------------------------------------------------
CONNECTION POINTS

  ;-----------------------------------------------------------
  FUNCTION GetTagText
    * Used to retrieve a number of tag text strings using an array of
      tag codes. This can be used on all standard tags that use text format.

    INPUT
      [word16_m:24]TagNumber   tagCodes

    OUTPUT
      [word16_m:24]TagText     values

    * Precondition
      none

    * Postcondition
      Return all tags found (may be less than requested if some
      requested codes are undefined). Check state code to verify
      validity of values and numbers to check existence of tags.
```

D.17.7 Interface PCCTagStream

```
INTERFACE PCCTagStream
  * This interface contains additional functionality to PCCTagDatabase
    to read a defined stream address.

  * Revision History
    990831  1.0 IEC issue
    981201  A  Added stream

  VERSION     1.0
  DATE        1999-08-31
```

```
  RESPONSIBLE IEC TC80/WG6

USAGE
  * Use together with PCCTagDatabase

;-------------------------------------------------------------
REQUIRED DOCUMENTATION

  TAGLIST    ; List of tag names supported by the interface. The list
             ; should preferably contain engineering units etc.

;-------------------------------------------------------------
REFERENCES
  General
  TagData

;-------------------------------------------------------------
CONNECTION POINTS

  ;-----------------------------------------------------------
  FUNCTION GetTagStream
    * Used to retrieve one tag stream.  The client supplies a stream
      address and the server, if it accepts the tag, is expected
      to try  to connect to the address after completing the call. The
      client shall have established the listening address prior to the
      call. The timeout is the maximum delay before the client
      should expect a conenction to be made.

    * The server will send data as soon as the connection has been
      established and will close the link when the last octet has been
      sent.

    INPUT
      TagNumber      tagCode
      address_m      address   ; Address of TP network
      word32_m       nnn       ; Node address
      TPSInstance    port      ; Additional port information

    OUTPUT
      StateCode      state     ; Current state or error
      word32_m       timeout   ; Timeout for connection attempt


    * Precondition
      Client has established listening address.

    * Postcondition
      Returns ok if tag is found and server is ready to send. The
      following state codes has special meaning:
      SC_NORMAL: Tag is ok and port will be connected to immediately.
      SC_NOT_AVAILABLE: Server is currently not available (e.g., other user)
```

D.17.8 Interface PCCTagNetsearch

```
INTERFACE PCCTagNetsearch
  * This interface contains additional functionality to search network
    via broadcast for tags.

  * Revision History
    1999-08-31 1.0 ojr, SINTEF: For IEC
    981201  A  First

  VERSION 1.0
  DATE 1999-08-31
  RESPONSIBLE IEC TC80/WG6

USAGE
  * Use together with PCCTagDatabase. Most users of the interface
    should define two interfaces: One for sending (CONNECT) and one
    for receiving (ACCEPT). Note that they are specified to operate on
    the Anonymous Broadcast MAU address ABCM1.

;-------------------------------------------------------------
REFERENCES
  General
```

```
  TagData

;-------------------------------------------------------------
CONNECTION POINTS

  ;-------------------------------------------------------------
  ANONYMOUS BROADCAST SendReq
    * Used to send a request for certain tag names. The functionality
      is similar to the general PCCTagDatabase.GetTagCodes function,
      except that only the MAU name of the keeper of the tag is
      returned and the return value must be retrieved through the
      GetAck MCP. Further investigations must be done on that MAU.

    OUTPUT
      TagKeyCode             keyType  ; Type of key used
      [word16_m:400]TagKey   key      ; Search key

  ;-------------------------------------------------------------
  ANONYMOUS BROADCAST GetAck
    * Used to receive a SendReq acknowledgement. Note that the search
      key is repeated.

    OUTPUT
      word8_m                status   ; Request status
      TagKeyCode             keyType  ; Type of key used
      [word16_m:400]TagKey   key      ; Search key
      int32_m                hits     ; Number of hits
      [32]char8_m            mauName  ; reporting MAU

    * Precondition
      Somebody sent a request.

    * Postcondition
      Returns the MAU name and the number of hits.
```

D.17.9 Interface PCCTagAttributes

```
INTERFACE PCCTagAttributes
  * This interface component contains additional functionality for reading
    tag attribute values from a data base.

  * Revision History
    1999-08-31 1.0 ojr, SINTEF: For IEC
    981201  C  Change name, moved data defs
    980813  B  Attribute ids valid for all tags
    980810  A  First

  VERSION 1.0
  DATE 1999-08-31
  RESPONSIBLE IEC TC80/WG6

USAGE
  * This interface allows the look up of attributes to tags. It
    requires the use of the CTagsDatabase interface for getting tag
    codes.

  * Attributes have a scope of the application they reside in. One
    attribute can, however, be valid for only one (or none)
    tags. There is a function to retreieve all attributes for one tag
    and there is a function to retrieve information on
    attributes. Some attributes can be given constant values for all
    applications.

;-------------------------------------------------------------
REFERENCES
  General
  TagData

;-------------------------------------------------------------
CONNECTION POINTS

  ;-------------------------------------------------------------
  FUNCTION GetTagAttrInfo
    Used to retrieve attribute codes for specified tags. NO_TAG
```

```
    returns all attributes.

    INPUT
      TagNumber    tag          ; For what tag
      int32_m      fromIndex    ; return more attributes

    OUTPUT
      [word16_m:40]TagAttrInfo  ; return values
      int32_m     nextIndex     ; signal more attributes

    * Precondition
      fromIndex shall be zero for first call. Tag code must be valid
      or NO_TAG.  If more attributes than can be returned by one call,
      fromIndex can be set to last return value of nextIndex in
      following calls.

    * Postcondition
      No values may be returned for invalid tags or attribute codes.
      Status field in each information block defines validity.
      nextIndex is non-zero if more attributes can be retrieved.

  ;-------------------------------------------------------------
  FUNCTION GetTagAttrValues
    Used to retrieve a number of attribute values, including alarm
    information, using an array of tag codes. This call will only
    return tags that have associated attribute values. State codes
    will give any error messages.

    INPUT
      [word16_m:40]TagNumber       tagCodes
      [word16_m:40]TagAttrNumber   attrCodes
      int32_m                      fromIndex

    OUTPUT
      [word16_m:96]TagAttrValues   values
      int32_m                      nextIndex

    * Precondition
      Any combination of tag codes and attribute codes can be
      used. The returned values are the intersection between the two
      groups (tag number AND attribute number). fromIndex is used if
      there are more values to be returned. It shall be zero on first
      call and can be nextIndex on subsequent calls.

    * Postcondition
      All valid combinations are returned. nextIndex is non-zero if
      more combinations are possible. In this case one can use
      repreated calls to retrieve all values.
```

D.17.10 Interface PCCTagSubscribe

```
INTERFACE PCCTagSubscribe
  * This interface contains the basic functionality for
    subscribing to tag values.

  * Revision History
    1999-08-31 1.1 ojr, SINTEF: For IEC
    990516  C  Max/min time, only one MCP
    981201  B  Changed name
    980826  A  First release, based on interface "CTagAlarm"

  VERSION 1.0
  DATE 1999-08-31
  RESPONSIBLE IEC TC80/WG6

USAGE
  * This interface allows subscribing to standard tag values. It
    requires the use of PCCTagDatabase and PCCTagDataSet.

;-------------------------------------------------------------
REFERENCES
  General
  TagData
```

```
;-------------------------------------------------------------
CONNECTION POINTS

  ;-------------------------------------------------------------
  INDIVIDUAL SUBSCRIBE TagSubscribe
    * Subscribe on values on a time base.

    INPUT
      TagSet       id          ; Set id
      Time         minInterval ; Min interval for updates
      Time         maxInterval ; Max interval for updates


    OUTPUT
      TagSet       id          ; Set id
      word16_m     status      ; Return status
      [word16_m:82]TagValue  value  ; Data

    * Precondition
      Set must be defined. Several subscriptions can be made on
      different sets. Set should be small enough for return value. The
      subscription principle is determined by timeouts:
On change: minInterval is zero
Watchdog: maxInterval non-zero
Limit messages: minInterval non-zero

    * Postcondition
      Initial call returns status code. The following are used:
      - BAD_SET (= 1), Illegal set code
      - SET_EMPTY (=2), No subscribe-able data in set
      - TOO_SHORT (=3), To small interval set
      - TOO_MUCH (=4), Set too large to send

      * The transaction should be cancelled if a non-zero status code
        is returned. This to avoid having pending transactions in the
        system.

      * Note that a unit down does not cause individual tag messages to
        be sent. Note also that changes in tag is value, alarm or state
        changes.

      * Subscription acknowledgement contains from one and upwards
        data entries. Several messages will be sent immediately after each
        other if there is not enough room in one message.
```

D.17.11 Interface PCCTagWrite

```
INTERFACE PCCTagWrite
  * This interface component contains the additional functionality for
    writing tag based data items to a data base.

  * Revision History
    1999-08-31 1.0 ojr, SINTEF: For IEC
    990516  C  Modified input to cover all tag state
    980826  B  Changed status code
    980810  A  First release, based on interface "TagDataExtended"

 VERSION 1.0
  DATE 1999-08-31
  RESPONSIBLE IEC TC80/WG6

USAGE
  * This interface allows writing tag values based on interface specific
    numeric tag codes. It is an add-on to the PCCTagDatabase
    interface. It may require user authentication.

;-------------------------------------------------------------
REFERENCES
  General
  TagData
  UserAuth

;-------------------------------------------------------------
CONNECTION POINTS
```

```
  ;-------------------------------------------------------------
  FUNCTION ModifyTagValue
   * Used to write and/or read values to a number of tags. The input
     field contains data for write or function type tags. The output
     field contains data for read or function type tags.

    INPUT
      [word16_m:32]TagValue   inValue  ; Input value

    OUTPUT
      UaStatus                authStatus ; Authorisation status
      [word16_m:32]TagValue   outValue   ; Any output

    * Precondition
      Input values must be defined for relevant tags. Values for
      non-input tags are ignored. All attributes (alarm, time, value)
      can be set.

    * Postcondition
      Return all tags with state code. Check state code for status on just
      written data. For read data the state code is as normal. The authState
      code is non-zero if the authentication failed (usually only for
      write)
```

D.17.12 Interface PCCTagAlarm

```
INTERFACE PCCTagAlarm
  * This interface component contains the basic functionality for
    handeling alarms on a tag name basis.

  * Revision History
    1999-08-31 1.0 ojr, SINTEF: For IEC
    981201  E  Change name, add acknowledgement
    980813  D  Only alarms, added individual subscribe
    980108  C  Added state update timeout exceeded
    971202  B  Removed tag info to TagData
    971111  A  First release, based on interface "Data"

  VERSION 1.0
  DATE 1999-08-31
  RESPONSIBLE IEC TC80/WG6

USAGE
  * This interface allows subscribing to alarms and acknowledgement of
    alarms. It requires a user authentication module in addition to
    the use of the basic PCCTagDatabase component.

;-------------------------------------------------------------
REFERENCES
    General
    TagData
    UserAuth

;-------------------------------------------------------------
CONNECTION POINTS

  ;-------------------------------------------------------------
  INDIVIDUAL SUBSCRIBE GetAlarms
    * Subscribe on new alarms on a given set of tags. Each client can
      subscribe on a different set, each client can also subscribe on
      a number of sets, each set representing one transaction.

    INPUT
      TagSetCode   id     ; Set id

    OUTPUT
      TagSetCode   id      ; Set id
      word16_m     oStatus ; Operation status
      [word16_m:40]TagAlarmValue  alarms  ; Alarms

    * Precondition
      Set must be defined. Several subscriptions can be made on
      different sets.
```

```
    * Postcondition
      Initial call returns only status code. The following are used:
      - BAD_SET (= 1), Illegal set code
      - SET_EMPTY (=2), No subscribable alarms in set
      - AUTHORISATION (=3)

      * The transaction should be canceled if a non-zero status code
       is returned. This to avoid having pending transactions in the
       system.

      * Subscription acknowledgement contains from one and upwards
       alarm entries.

  ;-------------------------------------------------------------
  FUNCTION AckAlarm
    * Acknowledge one alarm.

    INPUT
      TagNumber       id     ; The tag
      AlarmSequence   seq    ; The alarm

    OUTPUT
      UaStatus     aStatus   ; Authorisation status
      word16_m     oStatus   ; Operation status
      TagValue     value     ; New value for tag

    * Precondition
      Tag and sequence must be defined.

    * Postcondition
      Status code reports success (Zero) or failiure, value reports value
      after acknowledgement.
      - BAD_TAG (= 1), No such tag
      - BAD_SEQ (=2), No such alarm
      - BAD_AUTH (=3) Authorisation failed, check aStatus.
```

D.17.13 Interface PCCTagSet

```
INTERFACE PCCTagDataSet
  * This interface contains functionality for manipulating tag
    sets. The basic set can be defined by the GetTagCodes MCP and
    modified here. It can also be defined here.

  * Revision History
    1999-08-31 1.0 ojr, SINTEF: For IEC
    990516  C  Added define set to add
    981201  B  Changed name
    980813  A  First release

  VERSION 1.0
  DATE 1999-08-31
  RESPONSIBLE IEC TC80/WG6

USAGE
  * Used to inspect or modify tag sets.

  * All sets associated with one client is cleared when a client
    disconnects.

  * A set may be removed if it is empty.

  * A set can be cancelled by the only client that use it.

;-------------------------------------------------------------
REFERENCES
  General
  TagData

;-------------------------------------------------------------
CONNECTION POINTS

  ;-------------------------------------------------------------
  FUNCTION GetTagSet
```

```
    * Return tags in a tag set.

    INPUT
      TagSet      setCode       ; The set code
      int32_m     startIndex    ; Start returning records from this hit

    OUTPUT
      word8_m     status        ; Request status
      bool_m      more          ; More hits
      int32_m     endIndex      ; The hit index of the last code
      [word16_m:256]TagNumber  codes  ; Returned tags

    * Precondition

      * The startIndex entry shall be zero for first call on new
       search. To get more entries than can be returned by one call,
       startIndex shall be set to the previously returned endIndex
       and key kept constant for following calls.

    * Postcondition

      * status is zero for everything all right other error codes for
       status are:
       - BAD_SET (= 1), Illegal set code
       - NO_MORE (= 2), No more tags

      * more is true if there may be more hits. endIndex specifies the
       internal index of the next tag to be searched. Note that
       startIndex and endIndex is used to point into the server's
       internal data base and may not have any external
       interpretation.

  ;-------------------------------------------------------
  FUNCTION RemoveFromTagSet
    * Remove tags from a set

    INPUT
      TagSet      setCode       ; The set code
      [word16_m:256]TagNumber  codes  ; The codes to be removed

    OUTPUT
      word16_m    status        ; Request status
      int32_m     removed       ; Tags removed
      int32_m     left          ; Tags left

    * Precondition
      setCode must contain valid information. No tags mean delete whole
      set.

    * Postcondition
      + status is zero for everything all right other error codes for
       status are:
         - BAD_SET (= 1), Illegal set code
         - NOT_OWNER (= 2), Another application defined the set
         - FIXED_SET (=3), Set is not modifiable
      + removed and left counts the tags actually removed and the ones
       left in the set.

  ;-------------------------------------------------------
  FUNCTION AddToTagSet
    * Adds tags to a set or defines new set

    INPUT
      TagSet      setCode             ; The set code or null for new
      [word16_m:256]TagNumber  codes  ; The codes to be added

    OUTPUT
      TagSet      setCode       ; Set definition
      word16_m    status        ; Request status
      int32_m     added         ; Tags added

    * Precondition

      * setCode and codes must contain valid information. SetCode null
        means define new set.
```

```
    * Postcondition

      * status is zero for everything all right other error codes for
       status are:
      - BAD_SET (= 1), Illegal set code
      - NOT_OWNER (= 2), Another application defined the set
      - FIXED_SET (=3), Set is not modifiable

      * added counts the tags actually added to the set.
```

D.17.14 Interface PCCTagAttributeWrite

```
INTERFACE PCCTagAttributeWrite
  * This interface contains additional functionality for writing
    tag attribute values from a data base.

  * Revision History
    1999-08-31 1.0 ojr, SINTEF: For IEC
    981201  C  Changed name
    980813  B  Attribute ids valid for all tags
    980810  A  First

  VERSION 1.0
  DATE 1999-08-31
  RESPONSIBLE IEC TC80/WG6

USAGE
  * See PCCTagAttributes

;-------------------------------------------------------------
REFERENCES
  General
  TagData
  UserAuth

;-------------------------------------------------------------
CONNECTION POINTS

  REFERENCES
    PCCTagDatabase           ; All connection points

  ;-----------------------------------------------------------
  FUNCTION SetTagAttrValues
    Used to set a number of attribute values, including alarm
    information, using an array of tag codes.

    INPUT
      [word16_m:100]TagAttrValues  attrCodes

    OUTPUT
      UaStatus                   authStatus  ; Authentication status
      [word16_m:100]StateCode    result

    * Precondition
      Tags are identified with their values. The writing application
      should have authenticated the user and console.

    * Postcondition
      Authentication status returned overall. Individual status
      per attribute code position.
```

Annex E
(Normative)

Navigational interfaces

E.1 IEC 61162-1 relay function

The IEC 61162-1 standard contains requirements for data communication between maritime electronic instruments, navigation and radio-communication equipment when interconnected via an appropriate system.

Supporting one-way serial data transmission from a single talker to one or more listeners, the standard defines 63 different messages or so called *sentences* for data transfer. The purpose of this document is to supply appropriate data types, information classes and an interface for transmission of these sentences via the PISCES protocol. This Companion Standard document especially references to the chapter 6 *(content)* of IEC61162-1. It supports the transmission of all defined sentence types to ensure full compatibility.

The following clauses describe each of the interfaces and their connection points.

E.2 Interface PCCNMEAIn

The NMEAIn interface is used to connect a serial input stream of NMEA messages to the PISCES protocol. NMEA sentences can be read from one or more port addressed by the port number (<nn>). The interface provides the serially received sentences in two forms: via a SUBSCRIBE connection point, client can address all NMEA sentences (or of a particular sentence formatter) of a given port. By means of a FUNCTION connection point, clients can obtain particular sentences on demand.

The interface owns the following connection points:

E.2.1 READ NoOfPorts

Used to retrieve the number of available ports. Each port will have a respective subscribe-able Port_<nn> connection point.

E.2.2 FUNCTION GetPortDescription

Used to retrieve the description of a port. This is an informal text string, usually hard coded in the server.

E.2.3 FUNCTION NoOfSentences

Used to retrieve a number of supported sentences in GetSentence entry. Sentences and senders are used to select sentence for buffering.

E.2.4 FUNCTION GetListOfSentences

Used to retrieve the description of NMEA sentences supported by buffered GetSentence reads. Sentences and senders are returned in an array. Index in returned array starts at zero where user input startindex.

E.2.5 FUNCTION GetSentence

Used to retrieve a specific NMEA sentence from a specific port. The last received sentence with given formatter/sender is returned.

E.2.6 SUBSCRIBE Port_<nn>

This connection point is provided as an event-driven hook for reading NMEA messages from a specified port. A number of these connection points will be available. The number and names are defined by NoOfPorts. The names of the connection points are formed like "Port_01" where 01 identifies the port number. The description of the Port of the event is retrieved by GetPortDescription.

E.2.7 SUBSCRIBE Port_<nn>_<fmt>

This connection point is similar to the last one (Port_<nn>). However, it only provides sentences of the given formatter (<fmt>). For example, the following connection point might exist: "Port_02_VTG".

Note: These connection points are optional. Clients should use Port_<nn> if no appropriate connection point for the required formatter is available.

E.3 Interface PCCNMEAOut

This interface is used to write NMEA sentences to one or more serial ports. Clients can use a set of NON-ACKNOWLEDGE-WRITE connection points to write their data to a specified port.

The interface contains the following connection points:

E.3.1 READ NoOfPorts

Used to retrieve the number of available ports. Each port will have a respective subscribe-able Port_<nn> connection point.

E.3.2 FUNCTION GetPortDescription

Used to retrieve the description of a port. This is an informal text string, usually hard coded in the server.

E.3.3 NONACKED-WRITE Port_<nn>

This connection point is provided to write NMEA sentences to serial line port. A number of these connection points will be available. The number and names are defined by NoOfPorts. The names of the connection points are formed like "Port_01" where 01 identifies the port number. The description of the Port of the event is retrieved by GetPortDescription.

E.4 The NMEA related companion standard documents

E.4.1 The **NMEA data type description**

```
DATA TYPES NMEA
*  Definition of data types necessary to allow transmission of IEC 1162-1 (NMEA
0183)
   sentences over the PISCES protocol.

  * Sentence structure is defined in following references:

    * NMEA 0183 (1992), NMEA 0183 Standard for interfaceing marine
      electronics devices V 2.00, National Marine Electronics
      Association, Mobile AL, USA.

    * IEC 1162-1 DIS (1995), Maritime navigation and
      radiocommunication equipment and systems - Digital Interfaces -
      Part 1: Single talker and multiple listeners, refernce
      80/105/DIS.

  * Revision history:
    990831 2.1  For IEC (ojr, SINTEF)
    980824 V2   Adapted for PISCES project CS (mt/hg - ISSUS)
    951213 V1   Approved MCS. Typographical changes (ojr - SINTEF)
    951009 V1D  Committe draft for votes (ojr - SINTEF)
    950804 VB   Remove SentenceDescription, add Sender (Thor Vollset -
                Tordivel A/S)
    950626 VA   Change message to sentence, No to Instance (Thor
                Vollset - Tordivel A/S)
    950519      Created (Thor Vollset - Tordivel A/S)

  VERSION 2.1
  DATE 1999-08-31
  RESPONSIBLE IEC TC80/WG6


REFERENCES
  none

;-------------------------------------------------------------
INTERPRETATION Sentence OF [82]char8_m
   * Contains a NMEA message. Contents defined by NMEA 0183 standard
     or IEC 1162-1 standard.

;-------------------------------------------------------------
INTERPRETATION SentenceFormatter OF [3]char8_m
   * Used to identify NMEA 0183 message.

;-------------------------------------------------------------
INTERPRETATION Sender OF [2]char8_m
   * Used to identify the sender of a NMEA 0183 message.

;-------------------------------------------------------------
INTERPRETATION PortNo OF word16_m
   * Used to identify multiple ports on a server. Ports are numbered
     from 1 and upwards.

;-------------------------------------------------------------
INTERPRETATION Description OF [64]char8_m
   * General description string. May be null terminated.
     This type was defined in the data type definition MiTS.

;-------------------------------------------------------------
INTERPRETATION Boolean OF word8_m
   * A true/false type. Zero is false, non-zero true. For testing
     TRUE check that the type is not FALSE.
     This type was defined in the data type definition MiTS.

1 = TRUE
0 = FALSE

;-------------------------------------------------------------
```

```
INTERPRETATION BlockOk OF Boolean
   * This type says if contents of data block are all right.

* TRUE : contents are all right
  FALSE: contents are unreliable
```

- This type was defined in the data type definition MiTS.

E.4.2 Description of Interface PCCNMEAIn

```
INTERFACE PCCNMEAIn
  * This document contains the specification for the PISCES Companion
    Standard for receiving NMEA 0183 (IEC 1162-1) sentences over the
    PISCES protocol.

  * Revision history
    990831 2.1  For IEC (ojr, SINTEF)
    980824 V2   Adapted for PISCES project CS (mt/hg - ISSUS)
    951213 V1   Approved MCS. Typographical changes. (ojr - SINTEF)
    951009 V1D  Committee draft for vote. Added sender as index term
                for GetSentence. (ojr - SINTEF).
    950804 VB   Remove SentenceDescription add Sender (Thor Vollset
                - Tordivel A/S)
    950626 VA   Abstraction on port rather than sentence, Change
                message to sentence, No to Instance (Thor Vollset -
                Tordivel A/S).
    950519 V    Created (Thor Vollset - Tordivel A/S)

  VERSION 2.1
  DATE 1999-08-31
  RESPONSIBLE IEC TC80/WG6

USAGE
  The message management is performed by the connection points:
  - NoOfPorts, NoOfSentences and GetListOfSentences.
  Port_<nn> provides event driven connection. GetSentence is
  provided for the client if sporadic values are needed.

  * Port_<nn> re-transmits all sentences received on the specified
    port. NoOfPorts can be called to find the number of ports
    supported. <nn> is always two digits, i.e., from 01 to the
    number of ports supported.

  * NoOfSentences return zero if no sentences are buffered by the
    interface. If it returns non-zero GetListOfSentences can be used
    to get hold of the sentence formatters and senders supported by
    the interface. All supported sentences/senders will be allocated one
    buffer location so that the last received sentence of specified
    formatter/sender always is available. The valid field may indicate
    invalid message if, e.g., no message have been received after
    power up. The list of buffered sentences may be dynamically
    changed by the interface. No supported sentence shall, however, be
    removed.

  * Note that the interface does not check the syntax or semantics of
    the sentence prior to outputting it on the PISCES network.

;-----------------------------------------------------------
DATA TYPES

  REFERENCES
    NMEA VERSION 2.1

;-----------------------------------------------------------
CONNECTION POINTS

;-----------------------------------------------------------
  READ NoOfPorts
  * Used to retrieve the number of available ports. Each port will
    have a respective subscribe-able Port_<nn> connection point.

    OUTPUT
 PortNo  noOfPorts
```

```
    * Precondition
      none

    * Postcondition
      returns the number of serial ports available on this server.

    * Informal Explanation
      none

 ;-------------------------------------------------------------
  FUNCTION GetPortDescription
  * Used to retrieve the description of a port. This is an informal
    text string, usually hard coded in the server.

    INPUT
PortNo       noOfPort

    OUTPUT
Description  description
BlockOk      ok

    * Precondition
      1 <= noOfPort <= NoOfPorts

    * Postcondition
      if ok then message is returned
      else precondition is violated

    * Informal Explanation
      description is a free text description used for informal
      explanation of the device connected and sending NMEA 0183
      messages.

 ; -------------------------------------------------------------
  FUNCTION NoOfSentences
  * Used to retrieve a number of supported sentences in GetSentence
    entry. Sentences and senders are used to select sentence for
    buffering.

    INPUT
      PortNo    noOfPort

    OUTPUT
      word16_m       noOfSentences

     * Precondition
       none

     * Postcondition
       returns the number of NMEA 0183 sentences/senders buffered by
       the interface.

     * Informal Explanation
       returns zero if port is not present or no sentences are
       buffered.

  ; -------------------------------------------------------------
  FUNCTION GetListOfSentences
  * Used to retrieve the description of NMEA 0183 sentences supported
    by buffered GetSentence reads. Sentences and senders are
    returned in an array. Index in returned array starts at zero
    where user input startindex.

  INPUT
    PortNo    noOfPort
    word16_m       startindex
    word16_m       noOfElements

  OUTPUT
    [word16_m:20]Sender            sender
    [word16_m:20]SentenceFormatter formatter
    BlockOk                        ok

    * Precondition
```

```
      startindex < NoOfSentences
      startindex + noOfElements < NoOfSentences

    * Postcondition
      if ok then bit-wise indicated descriptions are returned
      else precondition is violated.

    * Informal Explanation
      startindex is numbered from zero.

; ------------------------------------------------------------
  FUNCTION GetSentence
  * Used to retrieve a specific NMEA 0183 sentence from a specific
    port. The last received sentence with given formatter/sender is
    returned.

    INPUT
      PortNo              noOfPort
      SentenceFormatter   sentence
      Sender              sender

    OUTPUT
      Valid          valid
      Message        message
      GlobalTime     time
      BlockOk        ok

    * Precondition
      message formatter/sender must have been returned by
      GetListOfSentences. port must be defined.

    * Postcondition
      if ok then message and time is returned. Validity defined by
      valid.
      else precondition is violated.

    * Informal Explanation
      The server shall store the last received message of all
      supported formatter/sender types in separate buffers.

 ; ------------------------------------------------------------
  SUBSCRIBE Port_<nn>
  * This connection point is provided as an event-driven hook for
    reading NMEA 0183 messages from a specified port. A number of
    these connection points will be available. The number and names
    are defined by NoOfPorts. The names of the connection points are
    formed like "Port_01" where 01 identifies the port number. The
    description of the Port of the instance is retrieved by
    GetPortDescription.

  OUTPUT
    Sentence   sentence

    * Precondition
      none

    * Postcondition
      When connection is established, the subscription can be started.

    * Informal Explanation
      The data will be transmitted from the server MAU when available.

; ------------------------------------------------------------
  SUBSCRIBE Port_<nn>_<fmt>
* This connection point is similar to the last one (Port_<nn>). However, it
  only provides sentences of the given formatter (<fmt>). Formatters can
  be any of those defined by IEC 1162-1 (table 5, approved sentence
  formatters). For example, the  following connection point may
  be part of the interface: "Port_02_VTG".

* Note: These connection points are optional. Clients should use Port_<nn> if
  no appropriate connection point for the required formatter is available.


  OUTPUT
```

```
    Sentence   sentence

    * Precondition
      none

    * Postcondition
      When connection is established, the subscription can be started.

    * Informal Explanation
      The data will be transmitted from the server MAU when available.
```

E.4.3 Description of Interface PCCNMEAOut

```
INTERFACE PCCNMEAOut
 * This document contains the specification for the PISCES Companion
   Standard for sending NMEA 0183 sentences over the PISCES protocol.

  * Revision history
    990831 2.1  For IEC (ojr, SINTEF)
    980824 V2   Adapted for PISCES project CS (mt/hg - ISSUS)
    951213 V1   Approved MCS. Typographical changes. (ojr - SINTEF).
    951009 V1D  Committee draft for vote. Also first version (ojr -
                SINTEF).
    951009      Based on interface NMEAIn


  VERSION 2.1
  DATE 1999-08-31
  RESPONSIBLE IEC TC80/WG6

USAGE
  The interface is similar to NMEAIn in that it supports some of the
  same  message management entries. The basic difference is that it
  allows a PISCES application to send NMEA 0183 sentences out on a
  serial port.

  * Port_<nn> sends all sentences written to the connection point.  It
    writes it on port nn, NoOfPorts can be called to find the number
    of ports supported. <nn> is always two digits, i.e., from 01 to
    the number of ports supported.

  * Note that the interface does not check the syntax or semantics of
    the sentence prior to outputting it on the serial line.

; ------------------------------------------------------------
DATA TYPES

  REFERENCES
    NMEA VERSION 2.1

;------------------------------------------------------------
CONNECTION POINTS

;------------------------------------------------------------
  READ NoOfPorts
   * Used to retrieve the number of available ports. Each port will
     have a respective write-able Port_<nn> connection point.

    OUTPUT
      PortNo  noOfPorts

    * Precondition
      none

    * Postcondition
      returns the number of serial ports available on this server.

    * Informal Explanation
      none

 ;------------------------------------------------------------
  FUNCTION GetPortDescription
   * Used to retrieve the description of a port. This is an informal
```

```
     text string, usually hard coded in the server.

    INPUT
      PortNo       noOfPort

    OUTPUT
      Description  description
      BlockOk      ok

    * Precondition
      1 <= noOfPort <= NoOfPorts

    * Postcondition
      if ok then message is returned
      else precondition is violated

    * Informal Explanation
      description is a free text description used for informal
      explanation of the device connected and sending NMEA 0183
      messages.

  ;-------------------------------------------------------
  NONACKED-WRITE Port_<nn>
  * This connection point is provided to write NMEA 0183 sentences to
    serial line port. A number of these connection points will be
    available. The number and names are defined by NoOfPorts. The
    names of the connection points are formed like "Port_01" where 01
    identifies the port number. The description of the Port of the
    instance is retrieved by GetPortDescription.

  INPUT
    Sentence   sentence

    * Precondition
      none

    * Postcondition
      Sentence written to port in the order that the connection point
      is written to.

    * Informal Explanation
      Note that several clients in principle can connect to the same
      port. This may be inhibited by the use of passwords.
```

E.4.4 Application Description

```
APPLICATION  PACNMEARelay
  * General MAU with serial line input or output ports that can read
    IEC 1162-1 sentences and make them available to the network or put
    them out to the serial ports.

  * The respective number of ports configured for output or input can
    be read through the interfaces.

  * Revision history
    990831 2.1  For IEC (ojr, SINTEF)
    980825 2.0 Second example, ISSUS

   VERSION 2.1
   DATE 1999-08-31
   RESPONSIBLE IEC TC80/WG6


USAGE
  * See referenced interfaces, This is an applciation acting as a
    relay between NMEA ports

REFERENCES

    PCCNMEAIn
    PCCNMEAOut
```

```
INTERFACES

  ACCEPT NMEAIn
    * This interface is used to read NMEA sentences buffered in the
    system. This application will automatically build a list of the
    sentences it has received and make them available through
    GetListOfSentences.

    INTERFACE COMPONENT PCCNMEAIn

  ACCEPT NMEAOut
    * This interface can output IEC 1162-1 sentences on configured
    serial line ports.

    INTERFACE COMPONENT PCCNMEAIn
```

_____